Conformance Testing of Temporal Role-Based Access Control Systems

Ammar Masood, Arif Ghafoor, *Fellow*, *IEEE*, and Aditya Mathur, *Member*, *IEEE*

Abstract—We propose an approach for conformance testing of implementations required to enforce access control policies specified using the Temporal Role-Based Access Control (TRBAC) model. The proposed approach uses Timed Input-Output Automata (TIOA) to model the behavior specified by a TRBAC policy. The TIOA model is transformed to a deterministic se-FSA model that captures any temporal constraint by using two special events *Set* and *Exp*. The modified W-method and integer-programming-based approach are used to construct a conformance test suite from the transformed model. The conformance test suite so generated provides complete fault coverage with respect to the proposed fault model for TRBAC specifications.

Index Terms—Role-based access control (RBAC), temporal role-based access control (TRBAC), finite-state models, timed input-output automata (TIOA), W-method, fault model, se-FSA transformation, integer programming (IP).

1 Introduction

Access control is one of the key security services used for information and system security. To control the time-sensitive activities in various applications such as workflow management systems and real-time databases, access control specifications are augmented with temporal constraints. As an example, workflow applications used in healthcare are required to enforce temporal access constraints to ensure the security of patient records [21]. One such constraint is to allow a doctor access to the patient record for only a specified duration. Role-based access control (RBAC) [11], which is particularly well suited for specifying access control policies and rules for any arbitrary organization-specific security model, has been extended with temporal constraints to enforce time-based access requirements [6], [14].

Here, we focus on conformance testing of software implementations required to enforce TRBAC policies. We refer to such an implementation as a "TRBAC system", or "access control implementation under test", or simply as ACUT. Our goal is to devise a scalable and effective conformance testing technique that provides complete fault coverage with respect to a TRBAC fault model. This fault model is developed using a mutation-based approach similar to the one used elsewhere [23].

A Timed Input-Output Automaton (TIOA), referred as $\mathrm{TRBAC_M}$, is used to model the expected behavior of an ACUT with respect to a TRBAC policy. A conformance test suite (CTS) is then generated from the $\mathrm{TRBAC_M}$ by first transforming it to an se-FSA [16], and then, using integer programming for determining the time stamps that satisfy the required temporal constraints in the access control policy by considering the sending of inputs and the monitoring of outputs at times that are an integral multiple of a minimum resolution. The ACUT is then executed against all elements of the CTS using the test architecture proposed in [15]. The proposed conformance testing approach only targets conformance testing of the ACUT with respect to a specific TRBAC policy. Additionally, functional testing is required to guarantee that ACUT will correctly enforce all TRBAC policies. As the set of all TRBAC policies is infinite, a representative subset of policies is used. Functional testing of ACUT is carried out as per the methodology presented in [22].

Contributions. 1) A fault model corresponding to TRBAC policy ($TRBAC_P$) specification; 2) a technique for modeling the expected behavior of TRBAC systems (ACUT) using TIOA; and 3) conformance testing strategy which provides complete fault coverage with respect to faults in the TRBAC fault model.

Organization. Section 2 outlines the sequence of steps in the proposed conformance testing approach. Section 3 defines the TRBAC policy specification, i.e., $TRBAC_P$, and presents an example used throughout the paper to explain the proposed conformance testing technique. The syntax and semantics of TIOA is also reviewed in Section 3. Section 4 discusses the "conformance relation" used to signify the connotation of conformance for ACUT. The conformance relation is used in Section 5 to study the fault model corresponding to any TRBAC policy specification. Construction of a TIOA-based model of any TRBAC policy is discussed in Section 6. Section 7 describes our procedure for the generation of CTS from this model. Two heuristics are briefly discussed in Section 8 to illustrate the application of state abstraction-based techniques to reduce the size of the TIOA model for any given policy. Section 9 briefly reviews the related work, and Section 10 summarizes the current work. A formal description of some construction procedures and the treatment of fault coverage are relegated to the appendixes, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41.

- *A. Masood is with the Department of Avionics Engineering, Institute of Avionics and Aeronautics, Air University, E-9 Islamabad, Pakistan. E-mail: ammar.masood@mail.au.edu.pk.*
- *A. Ghafoor is with the School of Electrical and Computer Engineering, Purdue University, 465 Northwestern Ave., West Lafayette, IN 47907. E-mail: ghafoor@ecn.purdue.edu.*
- *A. Mathur is with the Department of Computer Science, Purdue University, 250 N. University Street, West Lafayette, IN 47906. E-mail: apm@cs.purdue.edu.*

Manuscript received 7 Sept. 2006; revised 5 Nov. 2007; accepted 1 Apr. 2008; published online 21 July 2008.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0127-0906.
Digital Object Identifier no. 10.1109/TDSC.2008.41.

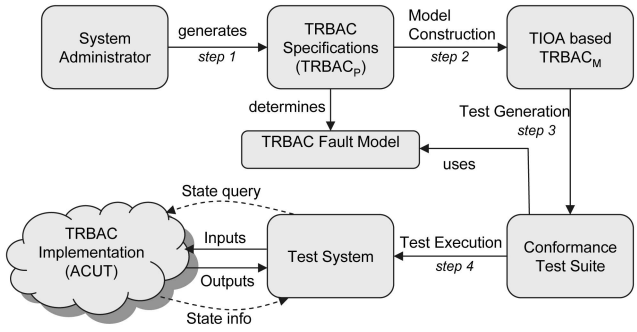

Fig. 1. Proposed approach for conformance testing of TRBAC Systems (ACUT).

2 Proposed Approach for Conformance Testing

The proposed approach for conformance testing of an ACUT is shown in Fig. 1. In step 1, a system administrator generates a TRBAC policy referred to as $TRBAC_P$. This policy is used to construct the expected behavior of the ACUT in step 2. This behavior is captured as a TIOA model referred to as $\mathrm{TRBAC_M}$. The test generator uses $\mathrm{TRBAC_M}$ as input and generates the CTS in step 3. A specific test system architecture, discussed in detail in Section 7.3, is used to execute the elements of the CTS against the TRBAC ACUT in step 4. The results of test execution are then compared with the expected behavior implied by $\mathrm{TRBAC_M}$ to validate ACUT conformance with respect to $TRBAC_P$.

Note that the four steps described in Fig. 1 deal with the testing of an ACUT with respect to only one TRBAC policy. In practical environments, one would expect, or at least desire, that while policies change, the ACUT does not. A functional testing procedure to check for the correctness of an ACUT against all policies is proposed elsewhere [22].

3 Background

3.1 Temporal RBAC Policy Specification

Temporal RBAC extends the RBAC model by time constraining a user's access to system resources. This is achieved by restricting the time duration of user-role activations, assignments, and permission-role assignments. A TRBAC specification also includes the rules for nontemporal user-role assignment (activation), separation of duty (SoD) constraints [2], role hierarchy semantics, and static/dynamic user (role) cardinality constraints. A formal definition of TRBAC policy specification follows.

Definition 3.1. ($\mathbf{TRBAC_P}$). *A* TRBAC *policy* $TRBAC_P$ *is a* $17\text{-}tuple$ $(U, R, Pr, Status, Permitted, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s, \Re)$, *where*

- $U, R,$ *and* Pr *are, respectively, the finite sets of users, roles, and permissions.*
- $Status = UR_{assign} \cup UR_{active} \cup PR_{assign}$ *is a set of status predicates partitioned as follows: 1)* UR_{assign}*:* $U \times R \rightarrow \{0,1\}$*, where a 1(0) indicates that the given user is assigned (not assigned) to the given role; 2)* UR_{active}*:* $U \times R \rightarrow \{0,1\}$*, where a 1(0) indicates that the given user has activated (not activated) the given role; and 3)* PR_{assign}*:* $Pr \times R \rightarrow \{0,1\}$*, where a 1(0) indicates that the given permission is assigned (not assigned) to the given role.*
- $$Permitted = UR_{canAssign} \cup UR_{canActivate} \cup PR_{canAssign}$$

  is a set of allowable predicates partitioned as follows: 1) $UR_{canAssign}$*:* $D_1 \subseteq U \times R \rightarrow$ *{1}, where the value of 1 indicates that the given user can be assigned to the given role; 2)* $UR_{canActivate}$*:* $D_2 \subseteq U \times R \rightarrow \{0,1\}$*, where a 1(0) indicates that the given user can activate (not activate) the given role; and 3)* $PR_{canAssign}$*:* $D_3 \subseteq Pr \times R \rightarrow$ *{1}, where the value of 1 indicates that the given permission can be assigned to the given role.*
- $\leq_A \subseteq R \times R$ *and* $\leq_I \subseteq R \times R$ *are, respectively, activation and inheritance hierarchy relations on roles.*
- $I = \{AS, DS, AC, DC, AP, DP\}$ *is a finite set of allowable input requests, where AS, DS, AC, DC, AP, DP are, respectively, Assign, Deassign, Activate, and Deactivate requests for user-role assignment and activation and Assign and Deassign for permission-role assignments. AS, AC, AP inputs optionally specify* $t \in Z^+$*, where* t *stipulates the maximum time duration of the corresponding assignment or activation and* Z^+ *denotes the set of nonnegative integers.*
- $S_u, D_u : U \rightarrow Z^+$ *are, respectively, static and dynamic cardinality constraints on* U*.*
- $S_r, D_r : R \rightarrow Z^+$ *are, respectively, static and dynamic cardinality constraints on* R*.*
- $SSoD, DSoD \subseteq 2^R$ *are, respectively, static and dynamic Separation of Duty (SoD) sets.*
- $S_s : SSoD \rightarrow Z^+$ *specifies the cardinality of the* $SSoD$ *sets.*
- $D_s : DSoD \rightarrow Z^+$ *specifies the cardinality of the* $DSoD$ *sets.*
- $\Re$ *is the rule set as in Definition 3.2.*

For clarity of presentation, a policy $TRBAC_P$ is simply referred as P unless noted otherwise. P is explicitly attached with each element of the above 17-tuple when there is a need to distinguish it from that of another policy. For example, $Status(P)$ and $Status(P')$ are the status predicates corresponding, respectively, to policies P and P'.

The activation hierarchy relation (A-hierarchy [24]) $r_i \leq_A r_j$ implies that a user u_k assigned to r_j is also able to activate r_i without being assigned to it, i.e., $UR_{assign}(u_k, r_i) = 0$. The inheritance hierarchy relation (I-hierarchy [24]) $r_i \leq_I r_j$ means that a permission p_k assigned to r_i is also accessible by r_j without being directly assigned to it, i.e., $(p_k, r_j) \notin PR_{canAssign}$. The static (dynamic) cardinality of a user specifies the maximum number of roles it can be assigned to (can activate). Similarly, the static (dynamic) cardinality of each role specifies the maximum number of users who can be assigned to (can activate) this role.

The $SSoD$ ($DSoD$) [2] specifies the sets of roles to which users can only be simultaneously assigned (can simultaneously activate) provided such assignments (activations) do not violate the $SSoD$ ($DSoD$) set cardinality constraint, i.e., $S_s(SSoD)$ ($D_s(DSoD)$). $S_s(SSoD)$ ($D_s(DSoD)$) constrains the maximum number of roles to which a user can

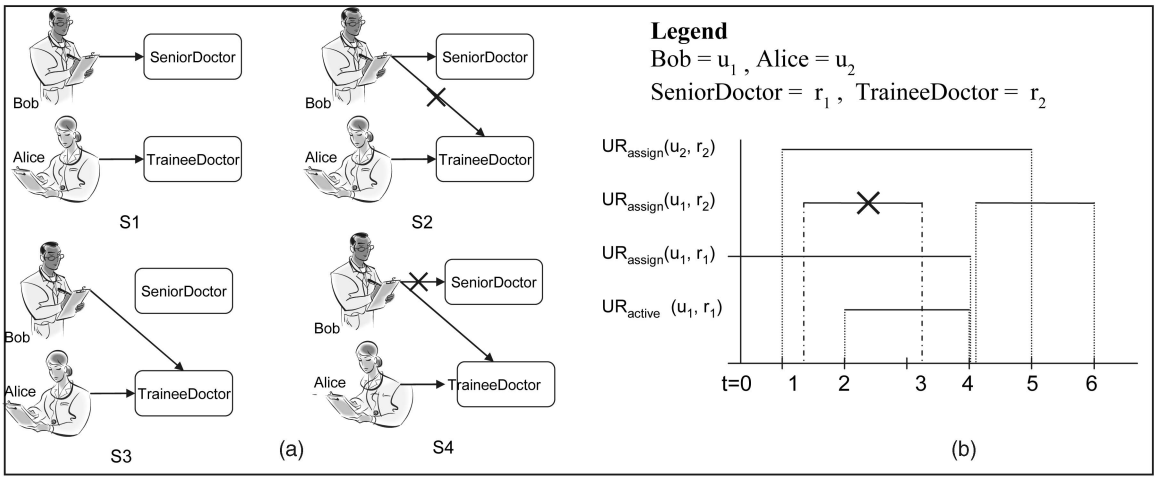

Fig. 2. (a) Various scenarios of user-role assignments allowed by domain TRBAC specification. (b) Effect of temporal and nontemporal constraints on user-role assignments and activation.

be simultaneously assigned (can simultaneously activate) in the given $SSoD$ ($DSoD$) set.

The access control decisions are guided by the formally specified rules in the rule set ($\Re$). The set of $Status$ and $Permitted$ predicates and the input requests $\{AS, DS, AC, DC, AP, DP\}$ in P are used to define these rules, which constrain the possible assignments and activations within the given TRBAC policy. The first six rules follow the syntax of first-order predicate formulae, whereas the last three are based on the temporal constraints and execution semantics of GTRBAC [14]. For clarity, predicates such as

$$< \left(\sum_{R} UR_{assign}(u, r_i), S_u(u) \right)$$

are written as $\sum_R UR_{assign}(u, r_i) < S_u(u)$. Next, we present the syntax and semantics of the rules in $\Re$.

Definition 3.2 ($\Re$). *Set*

$$\Re = \{\gamma_{urAssignCard}, \gamma_{urActiveCard}, \gamma_{urSSoD}, \gamma_{urDSoD}, \gamma_{urHier}, \gamma_{prHier}, \gamma_1, \gamma_2, \gamma_3\}$$

in Appendix IV, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41, describes system rules that dictate the access control decisions in a given $TRBAC_P$.

Example 1. Consider a simple TRBAC policy specification for a medical information system. There are two roles `SeniorDoctor` and `TraineeDoctor` (for simplicity, we refer these as r_1 and r_2, respectively) and two users Bob and Alice (referred to as u_1 and u_2, respectively). Fig. 2a shows the various basic user-role assignment scenarios allowed by the TRBAC policy of the domain. S1 scenario shows that both u_1 and u_2 are allowed to be assigned to r_1 and r_2, respectively. However, S2 indicates that if u_1 is already assigned to r_1, then it cannot be assigned to r_2 simultaneously. Thus, this represents the domain static SoD policy that a user cannot be simultaneously assigned to both `SeniorDoctor` and `TraineeDoctor` roles. Scenarios S3 and S4 are complementary to S1 and S2 as they indicate that if u_1 is initially assigned to r_2, then it cannot be simultaneously assigned to r_1.

Fig. 2b illustrates the temporal impact of various user-role assignment and activation inputs. The horizontal axis represents the value of global clock t. The lines above the horizontal axis, which show various status predicates, denote that corresponding user-role activation or assignment is valid for the time duration specified along the horizontal axis. The given $TRBAC_P$ is:

$$\begin{aligned} U = \{u_1, u_2\},\ R = \{r_1, r_2\},\ SSoD = \{S_1 = (r_1, r_2)\},\\ S_s(S_1) = 1,\ UR_{canAssign}(u_1, r_1) = UR_{canAssign}(u_1, r_2)\\ = UR_{canAssign}(u_2, r_2) = 1,\ S_u(u_1) = D_u(u_1) = 2,\\ S_u(u_2) = D_u(u_2) = 1,\ S_r(r_1) = D_r(r_1) = 1,\ S_r(r_2)\\ = D_r(r_2) = 2. \end{aligned}$$

Consider that in relation to scenario S1, at $t = 0, u_1$ is assigned to r_1 for 4 time units ($AS(u_1, r_1, 4)$ *issued at* $t = 0$), and u_2 is assigned to r_2 at $t = 1$ for same duration ($AS(u_2, r_2, 4)$ *issued at* $t = 1$), correspondingly the horizontal lines for status predicates $UR_{assign}(u_1, r_1)$ and $UR_{assign}(u_2, r_2)$ extends from 0 to 4 and from 1 to 5, respectively. Now if a request for u_1 activation of r_1 is made at $t = 2$ for three time units ($AC(u_1, r_1, 3)$ *issued at* $t = 2$), then by virtue of the duration constraint on the assignment of u_1 to r_1, the activation would also be terminated at $t = 4$ together with the removal of the assignment.

Considering the static SoD constraint, corresponding to scenario S2, on the concurrent assignment of a user to roles r_1 and r_2, it restricts u_1 ability to be assigned to r_2 before $t = 4$; because earlier than that rule γ_1 will be violated if such assignment is made ($\gamma_{urAssignCard}(u_1, r_2) \wedge \gamma_{urSSoD}(u_1, r_2)$ should be true but $\gamma_{urSSoD}(u_1, r_2)$ does not hold true at that time). Hence, the input request $AS(u_1, r_2, 2)$ cannot be granted before $t = 4$ but will be valid at/after $t = 4$.

3.2 Timed Input-Output Automata (TIOA)

The inclusion of temporal constraints in TRBAC requires precise modeling of real-time considerations which cannot be achieved by traditional finite-state machines. We, therefore, use timed automata [3], [13], in particular TIOA to model real-time constraints in a TRBAC specification. The syntax and semantics of TIOA is explained in detail next.

TIOA [13] are finite automata which partition the actions into input and output actions. Time is incorporated through the use of (real-valued) clocks; thus, TIOA is based on dense time semantics. A TIOA is a finite-state automaton with a finite set of locations and a set of labeled transitions. Each transition is labeled with an action which belongs to either the set of input actions or the set of output actions (hence, the name timed input-output automaton). The input and output actions begin with "?" and "!," respectively. The set of real-valued clocks C is used to specify timing constraints (known as guards) on the transitions. A transition is only executed if the associated guard satisfies the current value of clocks; moreover, a subset of clocks may be reset on executing a transition.

Definition 3.3. (TIOA). *A TIOA is a tuple* $A = \{L, l_0, I, O, C, T\}$ *where:*

- L *is a finite set of locations,* $l_0 \in L$ *is the initial location,* I *is a finite set of input actions, where each input action begins with "?,"* O *is a finite set of output actions, where each output action begins with "!," and* C *is a finite set of clocks.*
- $T \subseteq L \times (I \cup O) \times \Phi(C) \times 2^C \times L$ *is a set of transitions. A transition* $(l, \{?i, !o\}, g, R, l')$ *represents an edge from the location* l *to* l' *on input or output action* $?i$ *or* $!o$*. The guard* $g \in \Phi(C)$ *specifies the clock constraint which must be satisfied to enable execution of this transition. The set* $R \subseteq 2^C$ *gives the set of clocks which are reset to 0 on executing this transition.*

The transitions are assumed to be instantaneous in a TIOA. The term $\Phi(C)$ signifies the set of clock constraints specified using the clocks in C, where a guard $g \in \Phi(C)$ is defined by the grammar: $g := x \leq c | c \leq x | x < c | c < x | x = c | g \wedge g$. In this grammar, $x \in C$ and $c \in Z^+$ and Z^+ is the set of nonnegative integers. A clock valuation is a function $v : C \rightarrow \mathbb{R}_{\geq 0}$ that assigns a nonnegative real number to each clock in C. For a valuation $v, v + \delta (\delta \in \mathbb{R}_{\geq 0})$ denotes the valuation that assigns each clock $x \in C$ the value $v(x) + \delta$. For $Y \subseteq C, v[Y := 0]$ denotes the clock valuation for C which assigns 0 to each $x \in Y$ and agrees with v over other clocks. The set of all clock valuations is denoted by V_C. A valuation v satisfies a guard g if and only if g holds under v (it is represented as $v \approx g$). In the definition of TIOA, it is assumed that the domain of each clock $x \in C$ is bounded to $[0, C_x] \cup \{\infty\}$, where $C_x = max\{c | c \text{ is used in a } g \text{ over } x\}$.

Definition 3.4. (Semantics of TIOA). *The semantics of a TIOA* A *is defined by associating an infinite-state graph or Labeled Transition System (LTS)* $S_A = \{Q, \mathbb{R}_{\geq 0} \cup (I \cup O), \xrightarrow{a}\}$ *with* A*, where* $a \in \mathbb{R}_{\geq 0} \cup (I \cup O)$ *and:*

- Q *is the set of all states where each state is a pair* (l, v) *such that* l *is a location of* $A (l \in L)$ *and* v *is a clock valuation for* $C (v \in V_C)$*. The initial state of* S_A *is represented by* (l_0, v_0)*, where* $v_0(x) = 0$ *for all clocks* $x \in C$*.*
- *Edges of* S_A *are given by the relation* $\xrightarrow{a}$ *and are labeled with labels from the alphabet* $\mathbb{R}_{\geq 0} \cup (I \cup O)$*. There are two types of edges: discrete and timed edges. A discrete edge corresponds to a transition* (l, a, g, r, l') *of* A *and is represented as* $(l, v) \xrightarrow{a} (l', v[r := 0])$*, where* $a \in (I \cup O)$ *and* $v \approx g$*. For a state* (l, v) *and a time increment* $\delta \in \mathbb{R}_{\geq 0}$*, the timed edge* $(l, v) \xrightarrow{\delta} (l, v + \delta)$ *represents passing of time.*

Infinite states of S_A are by virtue of the infinite number of timed edges that can exist in S_A.

Definition 3.5. (Timed word). *A timed word over the alphabet* $\mathbb{R}_{\geq 0} \cup (I \cup O)$ *is a sequence* $w = (a_0, t_0), (a_1, t_1), \ldots, (a_k, t_k)$*, where each* $a_i \in \mathbb{R}_{\geq 0} \cup (I \cup O)$*, each* $t_i \in \mathbb{R}_{\geq 0}$*,* $0 \leq i \leq k$*, and the occurrence time* t_i *increases monotonically.*

Definition 3.6. (Run of A). *A* run *of* A *(starting from the initial location* l_0*) over a timed word* w *is a series*

$$(l_0, v_0) \xrightarrow{t_0} (l_0, v_0 + t_0) \xrightarrow{a_0} (l_1, v_1) \xrightarrow{t_1 - t_0} (l_1, v_1 + (t_1 - t_0)) \rightarrow \cdots \xrightarrow{a_k} (l_{k+1}, v_{k+1}),$$

where $v_i = v_{i-1} + (t_{i-1} - t_{i-2}), i > 1$.

The set of timed words accepted by A is denoted by $L(A)$ and it signifies the valid runs of A over the alphabet $\mathbb{R}_{\geq 0} \cup (I \cup O)$. For a state $s \in Q$ and a timed word w, we write $s \xrightarrow{w}$ iff $s \xrightarrow{w} s'$ for some $s' \in Q$. Time can progress in A iff the automaton is timelock-free [26], i.e., if every infinite run of S_A is *strongly non-Zeno*. The *non-Zeno* property ensures that A does not force its environment to provide an input by blocking time [18].

Definition 3.7. (Non-Zeno). *A is strongly non-Zeno iff for any state* $s \in Q$ *and any* $t \in \mathbb{R}_{\geq 0}$*, there is a timed output trace* $w = (a_0, t_0), (a_1, t_1), \ldots, (a_k, t_k)$*, where* $a_i \in (\mathbb{R}_{\geq 0} \cup O), 0 \leq i \leq k$ *such that* $s \xrightarrow{w}$ *and* $\sum_i (t_{i+1} - t_i) \geq t$.

A *timed automaton* would be *strongly non-Zeno* if it is ensured that at least one unit of time lapses in each of its loop [26], in case of TIOA A, this requirement is to hold for only such loops with actions from the set $(\mathbb{R}_{\geq 0} \cup O)$. It is important to verify that a TIOA is strongly non-Zeno as timelocks are modeling errors which should be resolved.

Next, we describe the conformance relation used in Section 5 to study the TRBAC fault model.

4 Conformance Relation

Let P be a TRBAC policy in effect, ACUT a correct implementation that enforces P and no other policy, and a possibly faulty ACUT′ required to enforce P. Let $Rq(up, r)$, $up \in (U \cup Pr)$, be a well-formed request such that $Rq \in I$ and $(up, r) \in (U \times R)$ for $up \in U$, and $(up, r) \in (Pr \times R)$ for $up \in Pr$. $Rq(up, r)$ is considered ill-formed when any one or more of the following conditions does not hold: $Rq \in I, up \in (U \cup Pr)$, and $r \in R$. The state of the ACUT with respect to P is the set $Status = UR_{assign} \cup UR_{active} \cup PR_{assign}$. All the status predicates of $Status$ are 0 at the start of ACUT execution. $Status$ changes in response to requests $Rq(up, r) \in I$. We write $Status'_{\text{ACUT}} = Status_{\text{ACUT}}[Rq(up, r)]$ to indicate that the updated status of ACUT in response to request Rq is $Status'_{\text{ACUT}}$ if the status prior to receiving $Rq(up, r)$ was $Status_{\text{ACUT}}$.

ACUT′ is said to conform *behaviorally* to ACUT with respect to policy P, under the following conditions.

TABLE 1
TRBAC Faults Due to Mutations of Elements of P

Structures Mutated	Possible Impact on TRBAC ACUT′ (Fault)
$UR_{canAssign}, S_u, S_r, SSoD, S_s$	UR1, UR2
$PR_{canAssign}, \leq_I$	PR1, PR2
$UR_{canActivate}, \leq_A, D_u, D_r, DSoD, D_s$	UA1, UA2
$\gamma_1, \gamma_2, \gamma_3$	All Temporal and Non-temporal faults

1. For all requests $Rq(up, r) \in I$, if $Status'_{\text{ACUT}} = Status_{\text{ACUT}}[R(up, r)]$, then $Status'_{\text{ACUT}'} = Status'_{\text{ACUT}} = Status_{\text{ACUT}'}[Rq(up, r)]$.
2. For all ill-formed requests $Rq(up, r)$, $Status_{\text{ACUT}}[Rq(up, r)]$ and $Status_{\text{ACUT}'}[Rq(up, r)]$ remain unchanged.
3. While there are no requests $\forall t \in \mathbb{R}_{\geq 0}$, $Status_{\text{ACUT}'} = Status_{\text{ACUT}}$.

Stated informally, the first two conditions imply that ACUT′ 1) assigns (deassigns) and activates (deactivates) a role only if such assignment (deassignment) and activation (deactivation) is allowable by the current policy; 2) assigns (deassigns) a set of permissions to (from) a role only if allowable by the current policy; and 3) ignores all ill-formed requests. The last condition implies that for all times, the $Status$ of ACUT and ACUT′ should remain unchanged in the absence of any input. Note that this condition does not imply that the state of an ACUT cannot change with time, rather if there is a change in the ACUT state, then correspondingly similar change is expected in the state of the conforming ACUT′.

5 TRBAC Fault Model

The TRBAC fault model is derived using a mutation-based approach [23]. The mutants $P' \neq P$ are obtained by applying the *set mutation* operators to the sets $Permitted, \leq_A, \leq_I, SSoD$ and $DSoD$ in P, *element modification* operators to the range of functions S_u, D_u, S_r, D_r, S_s, and D_s and *rule mutation* operators to the rules γ_1, γ_2, and γ_3 in $\Re(P)$. We consider three types of set mutation operators: modification of an element, addition of an element, and removal of an element. The semantics of element modification depends on the type of the element, which in case of another set implies recursive application of set mutation operators on the element. Only the application of *rule mutation* operator on a premise $i \in I$ varies in constructing the TRBAC fault model. As an $i \in I(TRBAC_P)$ can also specify $t \in Z^+$; therefore, in case of specification of duration, the rule mutation operator will replace t with $t+1$ or $t-1$. The details of application of other mutation operators are not given due to space limitation and can be found in [22].

Table 1 illustrates that the application of a mutation operator to P results in a policy P' which implies the possible presence of one or more faults in the ACUT′. As observed from Table 1 and Fig. 3, TRBAC faults can be broadly classified into two types: nontemporal and temporal faults. Nontemporal faults are related to user-role assignment, permission-role assignment, and user-role activation. Temporal faults are further categorized into hierarchical enforcement, duration widening, and duration restriction faults.

As shown in Fig. 3, each type of nontemporal fault is further categorized into two subcategories. Fault-type UR1 restricts an authorized user from being assigned to a role or leads to an unauthorized deassignment. Fault-type UR2 may lead to unauthorized role assignments. PR1 faults restrict a permission being assigned to an authorized role or cause an unauthorized deassignment. PR2 faults assign a permission to an unauthorized role. UA1 and UA2 faults are similar to UR1 and UR2 and impact role activation. The temporal faults are explained in detail next.

5.1 Hierarchical Enforcement Faults

A-hierarchy semantics allows user u assigned to a role r to activate a junior role $r' \leq_A r$ without any explicit assignment to r'. Similarly, a permission-role assignment can allow the automatic assignment of corresponding permission to all the senior roles by virtue of I-hierarchy semantics. P requires that the temporal constraints on explicit user role and permission-role assignments are also consistently enforced on implied activations or assignments (for clarity of discussion, we consider a user-role activation corresponding to a user-role assignment, also as implied activation, e.g., u_1r_1 activation corresponding to u_1r_1 assignment). However, errors in an ACUT′ may lead to erratic enforcement of temporal constraints on implied activations or assignments. Therefore, such faults are considered as hierarchical enforcement faults.

An instance of a hierarchical enforcement fault is illustrated in Fig. 4a. P considered in this instance is different from the one given in Example 1 as in this case, we consider that r_1 is senior to r_2 by virtue of A-hierarchy. Hence, u_1 assignment to r_1 also enables u_1 to activate r_2 without any explicit assignment to r_2. Furthermore, there is no SoD constraint, i.e., $SSoD = \{\}$. The temporal constraint that u_1r_1 assignment is restricted to a total duration of 4 time units (tus) also requires discontinuation of u_1r_2 activation at $t = 4$; however, the presence of hierarchical enforcement fault in the faulty ACUT′ permits u_1r_2 activation to continue beyond this time.

5.2 Duration Restriction Faults

The duration constraint on an event (user-role or permission-role assignment or user-role activation) requires that

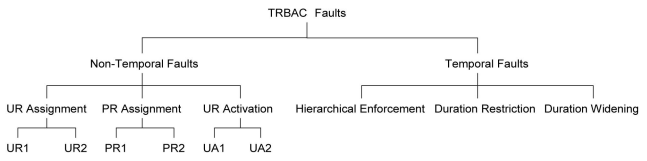

Fig. 3. A fault model for evaluating the effectiveness of tests for TRBAC implementations.

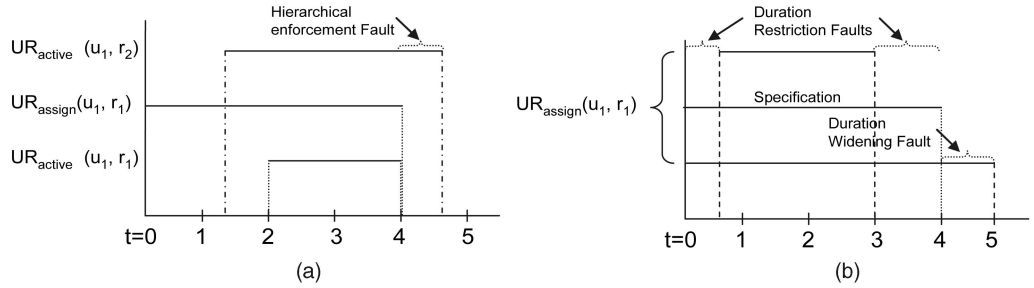

Fig. 4. Examples of temporal faults in an ACUT'. (a) A hierarchical enforcement fault. (b) Duration widening and restriction faults.

starting from the time, the event request is generated, e.g., $AS(u_1, r_1, 4)$ issued at $t = 0$ in Fig. 4b, the duration for which the corresponding event ($UR_{assign}(u_1, r_1) = 1$ in this case) is valid should be accurately enforced to be equal to the value specified by the constraint ($4tu$s in this case). The presence of duration restriction faults in an ACUT' would restrict the actual duration of the event to be less than the one specified in P. Duration restriction faults in an ACUT' can limit the actual duration of an event in two ways: by applying the deassignment/deactivation inputs before the time as required by P, or by delaying the required activations/assignments in the ACUT' as compared to P.

Fig. 4b illustrates both ways by which duration restriction faults affect the behavior of an ACUT'. It shows that although P requires $u_1 r_1$ assignment to be valid from $t = 0$ to $t = 4$, yet the same assignment is initially delayed in the ACUT', and then, deassignment occurs before $t = 4$ contrary to P. Duration restriction faults that delay the required activation/assignment can also be considered as nontemporal faults (of type UR1, UA1, or PR1).

5.3 Duration Widening Faults

As the name suggests, the impact of such faults is obvious in an ACUT'—the presence of duration widening faults would cause the duration of the associated event to be larger then the one allowed by P. The example in Fig. 4b demonstrates one such fault where the faulty behavior of ACUT' leads to an extension of the duration of $u_1 r_1$ assignment to more than specified by P.

We are justified in treating the hierarchical enforcement faults as duration widening or restriction faults. However, these faults are treated as a separate class because of their direct relationship with the constraints related to hierarchy semantics.

In the next section, we discuss the details of our proposed technique for capturing the expected behavior of a TRBAC ACUT.

6 Modeling the Expected Behavior of ACUT

As already mentioned, we use TIOA to model real-time constraints in P. For a conformance testing approach to be effective in detecting all types of faults that can exist in an ACUT (i.e., all the temporal and nontemporal faults in TRBAC fault model identified in Section 5), it is essential that the TRBAC model, referred as $\mathrm{TRBAC_M}$, should encapsulate all possible behaviors of the ACUT. This requires $\mathrm{TRBAC_M}$ to be able to capture the state of the ACUT in terms of all possible user-role assignments/activations and permission-role assignments that can exist in the ACUT, and the state transitions as valid actions determined by P.

There are two options in constructing the $\mathrm{TRBAC_M}$: 1) the requirements implied by P for user-role activations and assignments and permission-role assignments are treated in a single monolithic model and 2) divide the ACUT behavior into parts, and thus, describe the ACUT compositionally. We opted for the second option because of the convenience in reasoning the correctness of ACUT behavior with respect to P in compositional construction. Further, we consider it easier to extend $\mathrm{TRBAC_M}$ to model additional temporal constraints as they are added to TRBAC, if the ACUT behavior is described compositionally. We have considered compositional construction of $\mathrm{TRBAC_M}$ in terms of parallel composition of user-role and permission-role models, i.e., $\mathrm{TRBAC_M} = \mathrm{UR_M} \parallel \mathrm{PR_M}$. The parallel composition ($\parallel$) considered here is the one defined for parallel composition of timed automata in [26].

6.1 UR Model

The UR model ($\mathrm{UR_M}$) captures the desired response of an ACUT, as required by P, corresponding to all sequences of user-role assignments, deassignments, activations, and deactivations. $\mathrm{UR_M}$ thus encapsulates the conforming behavior of an ACUT where the behavior is captured with respect to user-role assignments and activations only. As indicated above, we use TIOA representation for $\mathrm{UR_M}$ to model temporal constraints on user-role assignments and activations. The construction of $\mathrm{UR_M}$ is considered in terms of parallel composition of basic UR models $\mathrm{UR_b}$s, i.e., $\mathrm{UR_M} = \mathrm{UR_{b1}} \parallel_{ur} \mathrm{UR_{b2}} \parallel_{ur}, \ldots, \mathrm{UR_{bk}}$, where k is the total number of $\mathrm{UR_b}$s. A $\mathrm{UR_b}$ is constructed corresponding to a user-role assignment; thus, the $\mathrm{UR_M}$ is comprised by constructing a $\mathrm{UR_b}$ corresponding to each user-role assignment possible in P, i.e., $k = |U \| R|$.

6.1.1 $\mathrm{UR_b}$ *Model*

There could be three types of $\mathrm{UR_b}$s: $\mathrm{UR_b^1}$, $\mathrm{UR_b^2}$, and $\mathrm{UR_b^3}$, where $\mathrm{UR_b^1}$ is the most general type and others are its special cases. In the subsequent discussion, a $\mathrm{UR_b}$, unless otherwise noted, refers to $\mathrm{UR_b^1}$. $\mathrm{UR_b}$ models are only constructed for those user-role pairs for which P provides an explicit assignment, i.e., $(u, r) \in D_1$ (Section 3.1).

A $\mathrm{UR_b}$ is composed corresponding to a specific user-role pair (u, r). In a $\mathrm{UR_b}(u, r)$, a pair of location variables is used to characterize the value of status predicates $UR_{assign}(u, r)$ and $UR_{active}(u, r)$. The A-hierarchy semantics is captured

by using location variables corresponding to $UR_{active}(u, r')$ for all such (u, r') pairs, where $r' \in (R' - \{r\})$ and R': $\{r' | r' \leq_A r\}$ is the set of all roles junior to r as per A-hierarchy semantics (r is also a member of R'). We assume that P is consistent [1], [20]; therefore, there are no explicit user-role assignments corresponding to the (u, r') pairs as such assignments are redundant. The value of location variables in a $\mathrm{UR_b}$ depicts the assigned/unassigned or active/inactive status of the corresponding user-role pair.

Assumptions. The deassignment and deactivation inputs (DS, DC) are assumed to be only initiated by the system (ACUT) as per the requirements of rules γ_1 and γ_2, respectively, and for clarity, these are modeled as output actions in the TIOA model. A user or a system administrator thus do not provide the DS and DC inputs before the system initiates the deassignments and deactivations automatically. An assignment/activation input leading to a particular user-role assignment/activation is assumed to be no longer available until the time the corresponding assignment/activation terminates. This is a reasonable assumption as, in practice, a user-role assignment/activation cannot be redone without a deassignment/deactivation first. These assumptions can be relaxed and have been kept primarily for simplicity of presentation. Relaxation of these assumptions would increase model complexity.

We have considered the outputs to be *urgent* [25], i.e., an output action transition is traversed soon after it gets enabled. The urgency assumption is required to correctly model the preemptive termination of user-role activations envisaged by γ_1 and the temporal constraints enforcement required by rule γ_2. We next formally define $\mathrm{UR_b}$ as a TIOA model.

Definition 6.1 ($\mathrm{UR_b}$)**.** *The basic user-role model* (UR_b) *corresponding to a user-role pair* (u, r) *is a TIOA* $UR_b(u, r) = \{L, l_0, I, O, C, T\}$ *where:*

- $L = \{l_0, l_1, l_2, \ldots, l_t\}$ *is a finite set of* t *locations such that*

$$l_s = \{UR_{assign}(u, r),\ UR_{active}(u, r),\ UR_{active}(u, r') | r' \in (R' - \{r\}),\ R' : \{r' | r' \leq_A r\}\}\ 1 \leq s \leq t$$

  is a set of status predicates. $l_0 = \{0, 0, \ldots, 0\}$ *is the initial location.*
- $I = \{?AS(u, r, t), ?AC(u, r', t) | r' \in R', R' : \{r' | r' \leq_A r\}\}$ *is a finite set of input actions.*
- $$O = \{!DS(u, r),\ !DC(u, r') | r' \in R',\ R' : \{r' | r' \leq_A r\}\}$$

  is a finite set of output actions.
- $C = \{x_1, x_2, \ldots, x_j\}$, *where* $j = |R'| + 1$ *is a finite set of clocks such that each clock* $x_i, 1 \leq i \leq j$ *corresponds to an input action.*
- $T \subseteq L \times (I \cup O) \times \Phi(C) \times 2^C \times L$ *is a set of transitions which are defined by the application of rules* γ_1 *and* γ_2 *on the input actions as explained in algorithm Construct*$\mathrm{UR_b}$, *given in Appendix I, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41.*

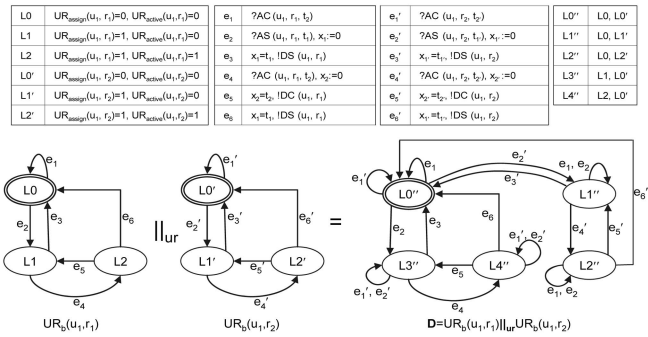

Fig. 5. $\mathrm{UR_b}(u_1, r_1)$ and $\mathrm{UR_b}(u_1, r_2)$ and their parallel composition.

It can be observed that $\mathrm{UR_b}$ satisfies the time progress requirement discussed in Section 3.2, i.e., it is *strongly non-Zeno* [26], as at least one unit of time would lapse in each loop of $\mathrm{UR_b}$ (the minimum value of t cannot be less than 1). Although we have considered the general case where all inputs are for temporal accesses, yet nontemporal inputs can also be easily modeled using TIOA. The $\mathrm{UR_b}$s for the user-role pairs (u_1, r_1) and (u_1, r_2) of Example 1 are illustrated in Fig. 5. The proof of Lemma 6.1 which shows the correctness of $\mathrm{UR_b}$ construction is given in Appendix I, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41.

Lemma 6.1 (Correctness of Construct$\mathrm{UR_b}$**).** *Given a* P *and a* (u, r) *pair, the algorithm Construct*UR_b *constructs a* UR_b *which correctly represents the application of rules from the rule set* $\Re(P)$ *to each state of ACUT corresponding to the user-role assignment/activations modeled by* UR_b.

As already mentioned in Section 4, the state of ACUT at a certain time is determined as the valuation of *Status* predicates at that time. Thus, each location of $\mathrm{UR_b}$ corresponds to a distinct state of ACUT.

A $\mathrm{UR_b}$ for (u, r) pair is constructed in isolation with other $\mathrm{UR_b}$s by considering that other then the user-role assignments/activations specified by the *Status* predicates corresponding to location variables of $\mathrm{UR_b}$, no other user-role assignments/activations or permission-role assignments exist in ACUT. As a result, not all the SoD and user/role cardinality constraint are modeled in $\mathrm{UR_b}$ (these constraints are fully imposed while constructing the $\mathrm{UR_M}$).

6.1.2 $\mathrm{UR_b^2}$ and $\mathrm{UR_b^3}$ Models

The $\mathrm{UR_b^2}$ and $\mathrm{UR_b^3}$ models, respectively, are constructed for all user-role pairs $(u, r) \in \{(U \times R) - (D_1 \cup D_4)\}$ and $(u', r') \in D_4$, where $D_4 = \{(u, r') \in (U \times R) - D_1 | (u, r) \in D_1, r' \leq_A r\}$. $\mathrm{UR_b^2}$'s are thus constructed corresponding to all those user-role pairs for which P does not provide information on both explicit assignment and implicit activation, whereas $\mathrm{UR_b^3}$'s correspond to such user-role pairs for which P does not

contain explicit assignment information but does allow implicit activation.

The UR_b^2 and UR_b^3 models are special cases of UR_b^1. The UR_b^2 model corresponding to a user-role pair (u, r) is a TIOA $\mathrm{UR}_\mathrm{b}^2(u, r) = \{L, l_0, I, O, C, T\}$, where

$$L = l_0 = \{UR_{assign}(u, r),\ UR_{active}(u, r)\},\ I = \{?AS(u, r, t),\ ?AC(u, r, t)\},\ O = C = \{\},$$

and

$$T = \{(l, ?AS(u, r, t), -, -, l), (l, ?AC(u, r, t), -, -, l)\}.$$

The UR_b^3 model corresponding to (u, r) is also a TIOA defined as $\mathrm{UR}_\mathrm{b}^3(u, r) = \{L, l_0, I, O, C, T\}$, where

$$L = l_0 = \{UR_{assign}(u, r)\},\ I = \{?AS(u, r, t)\}, O = C = \{\},$$

and $T = \{(l, ?AS(u, r, t), -, -, l)\}$.

Lemma 6.2 (Correctness of UR_b^2 and UR_b^3). *Given a P, $(u, r) \in \{(U \times R) - (D_1 \cup D_4)\}$ and $(u', r') \in D_4$, $UR_\mathrm{b}^2(u, r)$ and $\mathrm{UR}_\mathrm{b}^3(u', r')$ correctly represent the application of rules from the rule set $\Re(P)$ to each state of* ACUT *corresponding to the user-role assignment/activations modeled by UR_b^2 and UR_b^3, respectively.*

The UR_b^2 and UR_b^3 are constructed in isolation with any other UR_bs, and thus, it can be easily shown that they correctly represent the application of rules to each state of ACUT.

6.2 UR_M Construction

As discussed in Section 6.1, the UR_M corresponding to a P is constructed as a parallel composition of basic UR_bs of all types, i.e., $\mathrm{UR}_\mathrm{M} = \mathrm{UR}_\mathrm{b1} \parallel_{ur} \mathrm{UR}_\mathrm{b2} \parallel_{ur}, \ldots, \mathrm{UR}_\mathrm{bk}$, where $k = |U \| R| = |\mathrm{UR}_\mathrm{b}^1| + |\mathrm{UR}_\mathrm{b}^2| + |\mathrm{UR}_\mathrm{b}^3|$. We next define the binary operator $\parallel_{ur}$ for parallel composition of two UR_bs. UR_M is constructed by recursive application of $\parallel_{ur}$ operator on all the UR_bs. As UR_bs are *strongly non-Zeno*; therefore, it can be easily shown, by using the approach similar to Lemma 3 in [26], that their parallel composition UR_M would also be *strongly non-Zeno*.

Definition 6.2 ($\parallel_{ur}$). *Given two UR_bs, $A = \{L, l_0, I, O, C, T\}$ and $B = \{L', l_0', I', O', C', T'\}$, where $C \cap C' = \emptyset, I \cap I' = \emptyset$ and $O \cap O' = \emptyset$, the parallel composition $D = A \parallel_{ur} B$ is defined as $D = \{L_D \subseteq L \times L', (l_0, l_0'), I \cup I', O \cup O', C \cup C', T_D\}$ such that L_D and T_D are the smallest relations defined by the application of rules γ_1 and γ_2 on the input actions as explained in algorithm ParallelComposition in Appendix II, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41.*

The parallel composition $D = A \parallel_{ur} B$ is constructed by a constrained Cartesian product of the locations of A and B and the union of their clocks, inputs, and outputs. The locations L_D and transitions T_D of D are determined by the algorithm ParallelComposition. The total number of locations of D is less than or equal to the Cartesian product of L and L', i.e., $|L_D| \leq |L \times L'|$.

A TIOA constructed by parallel composition of two UR_bs can also be considered as another UR_b. The parallel composition $D = \mathrm{UR}_\mathrm{b}(u_1, r_1) \parallel_{ur} \mathrm{UR}_\mathrm{b}(u_1, r_2)$ corresponding to P of Example 1 is illustrated in Fig. 5. Lemma 6.3 formally shows (proof given in Appendix II, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41) that the parallel composition $D = A \parallel_{ur} B$ for the UR_bs A and B correctly models ACUT behavior with respect to the user-role assignments/activations modeled by A and B. D is constructed from the UR_bs A and B in isolation with all other UR_bs, i.e., the impact of other user-role assignments/activations in the ACUT is not considered in constructing D. The recursive application of $\parallel_{ur}$ ensures that the final UR_M correctly models ACUT with respect to all the user-role assignments/activations.

Lemma 6.3 (Correctness of ParallelComposition). *Given a P and two UR_bs A and B, the algorithm ParallelComposition constructs $D = A \parallel_{ur} B$ which correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT corresponding to the user-role assignments/activations modeled by A and B.*

Corollary 6.1 (Correctness of UR_M). UR_M *correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT with respect to all the user-role assignments/activations.*

The proof is simply based on the correctness of ParallelComposition shown by Lemma 6.3, as UR_M is constructed by recursive application of ParallelComposition on all the UR_bs.

6.3 PR Model

Permission-role model (PR_M) encapsulates the behavior of an ACUT with respect to permission-role assignments specified explicitly in P or implicitly allowed via I-hierarchy semantics. PR_M is constructed in a way similar to UR_M by considering parallel composition of basic permission-role models (PR_bs), i.e.,

$$\mathrm{PR}_\mathrm{M} = \mathrm{PR}_\mathrm{b1} \parallel_{pr} \mathrm{PR}_\mathrm{b2} \parallel_{pr}, \ldots, \mathrm{PR}_\mathrm{bj},$$

where $j = |D_6| + |D_3|$ is the total number of PR_bs, such that

$$D_6 = \{(p, r) \in (P \times R) | (p, r) \notin (D_3 \cup D_5)\}$$

and

$$D_5 = \{(p, r') \in (P \times R) - D_3 | (p, r) \in D_3, r \leq_I r'\}.$$

Set $(D_3 \cup D_5)$ corresponds to all such permission-role pairs for which assignments are either explicitly specified in P or implicitly permitted via I-hierarchy semantics, whereas the set D_6 represents such permission-role pairs for which P does not contain any assignment information.

6.3.1 PR_b Model

There could be two types of PR_bs: PR_b^1 and PR_b^2, where PR_b^1 is the most general type. In the subsequent discussion, unless otherwise noted, PR_b refers to PR_b^1. A PR_b is comprised corresponding to a specific permission-role pair (p, r). In the absence of roles senior to r by virtue of I-hierarchy semantics, PR_b would be very simple as only one location variable is used to characterize the value of status predicate $PR_{assign}(p, r)$. The I-hierarchy semantics are captured by using location variables corresponding to $PR_{assign}(p, r')$ for all such (p, r') pairs where $r' \in (R' - \{r\})$

and $R' : \{r' | r \leq_I r'\}$ is the set of all roles senior to r as per I-hierarchy semantics (r is also a member of R'). The value of location variables in $\mathrm{PR_b}$, therefore, depicts the assigned/unassigned status of the corresponding permission-role pair. Next, we formally define $\mathrm{PR_b}$ as a TIOA model.

Definition 6.3 ($\mathrm{PR_b}$)**.** *The basic permission-role model* ($\mathrm{PR_b}$) *corresponding to a permission-role pair* (p, r) *is a TIOA* $\mathrm{PR_b}(p, r) = \{L, l_0, I, O, C, T\}$ *where:*

- $L = \{l_0, l_1\}$ *is a set of two locations such that*

  $l_s = \{PR_{assign}(p, r') | r' \in R', R' : \{r' | r \leq_I r'\}\} s \in \{0, 1\}$

  is a set of status predicates. $l_0 = \{0, 0, \ldots, 0\}$ *is the initial location.*
- $I = \{?AP(p, r', t) | r' \in R', R' : \{r' | r \leq_I r'\}\}$ *is a finite set of input actions.*
- $O = \{\wedge(!DP(p, r') | r' \in R', R' : \{r' | r \leq_I r'\})\}$ *is a set of single-output action.*
- $C = \{x_1\}$ *is a set of single clock* x_1 *which corresponds to the input action* $?AP(p, r, t)$.
- $T \subseteq L \times (I \cup O) \times \Phi(C) \times 2^C \times L$ *is a set of transitions, defined by the application of rule* γ_3 *on inputs, as explained in the algorithm Construct*PR_b*, given in Appendix III, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41.*

While constructing a $\mathrm{PR_b}$, it is assumed that the deassignment input is only initiated by the system (ACUT) as per the requirements of rule γ_3, and thus, a system administrator does not provide these inputs before the system automatically initiates the deassignments. $\mathrm{PR_b}$ also satisfies the time progress requirement as it is *strongly non-Zeno*. The proof of Lemma 6.4 which shows the correctness of $\mathrm{PR_b}$ construction is in Appendix III, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.41.

Lemma 6.4 (Correctness of Construct$\mathrm{PR_b}$**).** *Given* P *and* (p, r), *Construct*PR_b *constructs a* $\mathrm{PR_b}$ *that correctly represents the application of rules from the rule set* $\Re(P)$ *to each state of ACUT corresponding to the permission-role assignments modeled by* $\mathrm{PR_b}$.

$\mathrm{PR_b^2}$ models are constructed for all the permission-role pairs $(p, r) \in D_6$ for which P does not provide both the explicit and implicit assignment information. A $\mathrm{PR_b^2}$ model corresponding to a permission-role pair (p, r), being a special case of $\mathrm{PR_b^1}$, is a TIOA $\mathrm{PR_b^2}(p, r) = \{L, l_0, I, O, C, T\}$, where $L = l_0 = \{PR_{assign}(p, r)\}, I = \{?AP(p, r, t)\}, O = C = \{\}$, and $T = \{(l, ?AP(p, r, t), -, -, l)\}$. The correctness of $\mathrm{PR_b^2}$ is simple to observe.

6.4 $\mathrm{PR_M}$ Construction

As already mentioned, $\mathrm{PR_M}$ is obtained as parallel composition of $\mathrm{PR_b}$s of both types, i.e.,

$$\mathrm{PR_M} = \mathrm{PR_{b1}} \parallel_{pr} \mathrm{PR_{b2}} \parallel_{pr}, \ldots, \mathrm{PR_{bj}},$$

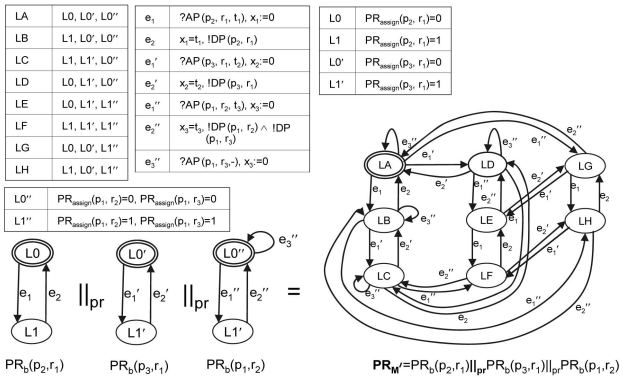

Fig. 6. Example of parallel composition of three $\mathrm{PR_b}$s.

where $j = |D_6| + |D_3| = |\mathrm{PR_b^1}| + |\mathrm{PR_b^2}|$. The binary operator $\parallel_{pr}$ is defined next for parallel composition of two $\mathrm{PR_b}$s. $\mathrm{PR_M}$ is constructed by recursive application of $\parallel_{pr}$ operator on all the $\mathrm{PR_b}$s. As $\mathrm{PR_b}$s are *strongly non-Zeno*; therefore, their parallel composition $\mathrm{PR_M}$ would also be *strongly non-Zeno* ([26, Lemma 3]).

Definition 6.4 ($\parallel_{\mathbf{pr}}$)**.** *Given two* $\mathrm{PR_b}$s $A = \{L, l_0, I, O, C, T\}$ *and* $B = \{L', l_0', I', O', C', T'\}$, *where* $C \cap C' = \emptyset, I \cap I' = \emptyset$, *and* $O \cap O' = \emptyset$, *the parallel composition* $D = A \parallel_{pr} B$ *is defined as* $D = \{L \times L', (l_0, l_0'), I \cup I', O \cup O', C \cup C', T_D\}$ *such that* T_D *is the minimum set of composite transitions formed by interleaving of individual transitions of* A *and* B. *Corresponding to a transition pair* (e_1, e_2), *where*

$$e_1 = (l_s, \{?i, !o\}, g, R, l_t) \in T$$

and

$$e_2 = (l_s', \{?i, !o\}', g', R', l_t') \in T',$$

two composite transitions $e_1' = ((l_s, l_s'), \{?i, !o\}, g, R, (l_t, l_s'))$ *and* $e_2' = ((l_s, l_s'), \{?i, !o\}', g', R', (l_s, l_t'))$ *will be added to* T_D *by virtue of interleaving of* e_1 *and* e_2.

The correctness of $\parallel_{pr}$ comes easily from Definition 6.4. As a corollary, it is simple to argue the correctness of $\mathrm{PR_M}$. An example of parallel composition of three $\mathrm{PR_b}$s is illustrated in Fig. 6. The P in Fig. 6 corresponds to Example 1 with an added role `ResidentDoctor` (r_3) senior to r_2 in the I-hierarchy. Assume that there are three permissions $p_1 =$ `Update patient record`, $p_2 =$ `Erase patient record`, and $p_3 =$ `Create patient record`, which can be assigned to roles r_1 and r_2. Consider that P specifies that p_2 and p_3 can be assigned to r_1, whereas only p_1 can be assigned to r_2. There would be three $\mathrm{PR_b^1}$ models tallying with p_2r_1, p_3r_1, and p_1r_2 explicit permission-role assignments in P. $\mathrm{PR_b}(p_1, r_2)$ illustrates that by virtue of I-hierarchy, p_1r_2 assignment would also automatically lead to p_1r_3 assignment without requiring any other input. The $\mathrm{PR_{M'}}$ constructed as a parallel composition $\mathrm{PR_b}(p_2, r_1) \parallel_{pr} \mathrm{PR_b}(p_3, r_1) \parallel_{pr} \mathrm{PR_b}(p_1, r_2)$ is also shown in Fig. 6. Note that $\mathrm{PR_M}$ would be constructed by parallel composition of $\mathrm{PR_{M'}}$ with all $\mathrm{PR_b^2}$ models, i.e., $\mathrm{PR_M} = \mathrm{PR_{M'}} \parallel_{pr} \mathrm{PR_b^2}(p_1, r_1) \parallel_{pr} \ldots \parallel_{pr} \mathrm{PR_b^2}(p_3, r_3)$.

Theorem 6.1 formally shows the correctness of $\mathrm{TRBAC_M} = \mathrm{UR_M} \parallel \mathrm{PR_M}$.

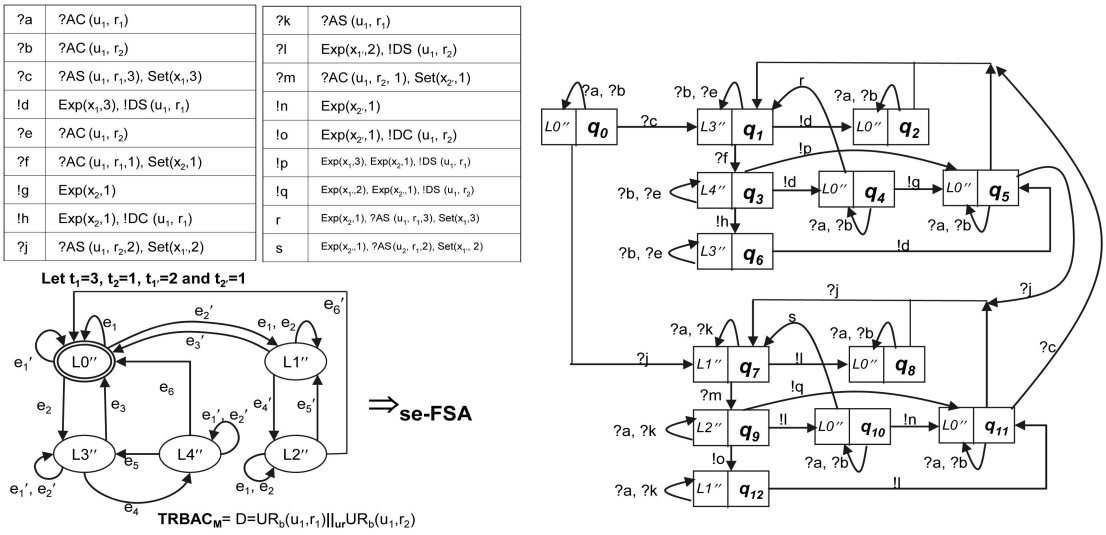

Fig. 7. se-FSA transformation of D, i.e., se-$\mathrm{TRBAC_M}$.

Theorem 6.1 (Correctness of $\mathrm{TRBAC_M} = \mathrm{UR_M} \parallel \mathrm{PR_M}$). *$\mathrm{TRBAC_M}$ correctly represents the application of rules from the rule set $\Re(P)$ to each state of* ACUT *with respect to all the user-role assignments and activations and permission-role assignments in P.*

Proof of Theorem 6.1 is based on the correctness of $\mathrm{UR_M}$ and $\mathrm{PR_M}$ shown earlier, as the parallel composition $\mathrm{UR_M} \parallel \mathrm{PR_M}$ does not cause any violation of the rules from the rule set $\Re(TRBAC_P)$.

We have already presented the TRBAC fault model in Section 5 and have now completed a description of the technique for the construction of $\mathrm{TRBAC_M}$ for a given TRBAC policy specification. In the next section, we focus on a procedure for the generation of the conformance test suite from $\mathrm{TRBAC_M}$ that provides complete coverage with respect to the faults in the proposed TRBAC fault model.

7 Test Generation from TRBAC Model ($\mathrm{TRBAC_M}$)

Key steps in the proposed technique for construction of conformance test suite from $\mathrm{TRBAC_M}$ are as follows: 1) transform $\mathrm{TRBAC_M}$ into se-FSA (se-$\mathrm{TRBAC_M}$) using the procedure in [16]; 2) construct the testing tree (Tr) corresponding to the se-$\mathrm{TRBAC_M}$; and 3) generate the conformance test suite. The conformance test suite is then executed against the ACUT using the test architecture in [15].

7.1 Transformation of $\mathrm{TRBAC_M}$ into se-$\mathrm{TRBAC_M}$

It is important to note that the semantic graph S_A of a TIOA A (Section 3.2), which encapsulates the information about all the accepting runs of A, is of infinite size. The primary purpose of se-FSA transformation is to capture the timed semantics of A by using a Finite-State Automaton (FSA). The se-FSA transformation converts a TIOA into an *equivalent* finite-state automaton which in addition to the events of the TIOA has the two special types of events: *Set* and *Exp* that model setting and expiring of clocks, respectively. The se-FSA is *equivalent* to its corresponding TIOA, as shown in [16], in the sense that both specify the same order and timing constraints of events. We omit the finer details of se-FSA transformation and refer the interested to [16] for the complete algorithm.

A se-FSA is an FSM $S_E = (Q, q_0, X, Y, \delta, O)$, where Q is a finite set of states, $q_0 \in Q$ a unique initial state, X and Y, respectively, the input and output alphabets, $\delta : Q \times X \rightarrow Q$ the state transition function, and $O : Q \times X \rightarrow Y$ the output function. A member such as Q of S_E is referred as $Q(S_E)$. The salient features of se-FSA transformation of $\mathrm{TRBAC_M}$ are illustrated by considering a simpler version of Example 1 with the single user u_1. For this case, $\mathrm{TRBAC_M} = \mathrm{UR_M} = D$. We have adapted the se-FSA procedure of [16] to handle urgent outputs in $\mathrm{TRBAC_M}$. In general, the se-FSA transformation of a deterministic TIOA may result in a nondeterministic FSM; however, the se-FSA corresponding to $\mathrm{TRBAC_M}$ would always be deterministic because $\mathrm{TRBAC_M}$ considers that urgent outputs and any clock resets in it are only associated with input actions.

The se-FSA transformation of $\mathrm{TRBAC_M}$, i.e., se-$\mathrm{TRBAC_M}$ of simplified Example 1, is illustrated in Fig. 7. It can be observed that there are three types of events in se-$\mathrm{TRBAC_M}$: input events corresponding to input actions and/or clock resets in $\mathrm{TRBAC_M}$, output events corresponding to output actions in $\mathrm{TRBAC_M}$ and/or clock expirations, and complex events occurring as combination of the previous two. The duration constraints specified in $\mathrm{TRBAC_M}$ become explicit in the se-$\mathrm{TRBAC_M}$ as all the input actions, associated with transitions other than self loops, would cause setting of a clock whose expiry will result in the corresponding deactivation/deassignment.

7.2 Construction of Testing Tree

As mentioned, se-$\mathrm{TRBAC_M}$ is deterministic and has a finite number of states. Therefore, we can use any test generation technique for deterministic FSAs to construct the conformance test suite. For our test generation for TRBAC ACUT, we used the W-method [9] due to its proven fault coverage. Tests are generated in this method by concatenating the test sequences obtained from the testing tree (Tr) with the determined state characterization set referred to as the W set. We assume the existence of reliable methods in the ACUT that can be used to directly query the current state. Hence, the W set is not required and the test set can be generated

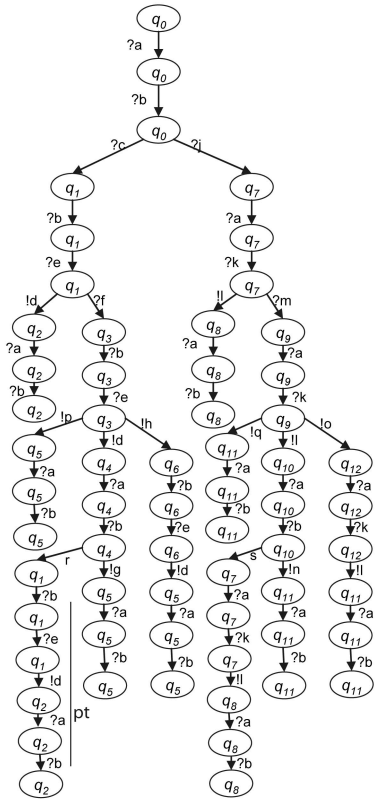

Fig. 8. Tr for se-$\mathrm{TRBAC_M}$ of Fig. 7.

directly from the testing tree. This assumption is not very restrictive and has also been used in the Binder round-trip method [7] for class testing in object-oriented programs. By virtue of state observability, the test suite is directly constructed from Tr. When the states are not observable, then using the procedure given in [15], the se-$\mathrm{TRBAC_M}$ can be made input complete and W set can be determined. The test suite generation in this case would be more complex and the test suite size would be much larger.

Note that state observability here implies the ability to determine the valuation of $Status$ predicates in the ACUT, which actually corresponds to location observability. In se-$\mathrm{TRBAC_M}$, this could cause a problem in detecting such transfer, missing state, and extra state faults where the correct and incorrect destination states correspond to the same location but different range of clock variables. However, as discussed later in Section 7.5, by virtue of the structure of $\mathrm{TRBAC_M}$ and its se-$\mathrm{TRBAC_M}$ transformation, transfer, missing state, and extra state faults leading to incorrect states corresponding to same location as of the correct state would always lead to output faults which can be detected with only location observability.

While constructing Tr from the FSM if the added node at depth k is the same as some other node at depth $i, i \leq k$, then that node is terminated with no further edge out of it [9]. For our conformance test suite, we considered a modified Tr to reduce the total number of test sequences by including the repeated nodes explicitly in the path only when they are encountered for the first time (corresponding to a specific event). Note that the paths in Tr still form a *transition cover set* as required by the W-method [9]. A Tr corresponding to se-$\mathrm{TRBAC_M}$ of Fig. 7 is illustrated in Fig. 8.

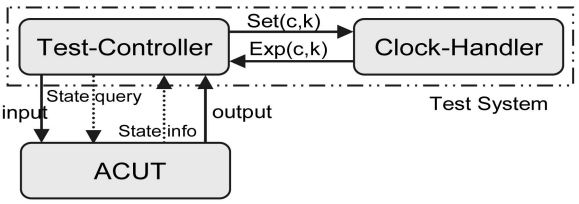

Fig. 9. Structure of test system (test harness) [15].

As the finite sequences accepted by an se-FSA should terminate in a state without outgoing *Exp* events (accepting state) [16], the terminals in a testing tree have to be from among the set of accepting states. Hence, if at any level in the Tr, a nonaccepting node is repeated, the corresponding path is not terminated until the time the terminal node is an accepting state. We suggest that the shortest path among all paths from a repeating nonaccepting node to any of the accepting states be used for this path augmentation. The path "pt" in the Tr of Fig. 8 depicts that although q_1 is repeated along this path for the same events "?b" and "?e" yet the path is not terminated because q_1 is not an accepting state.

7.3 Generation of Conformance Test Suite (CTS)

The se-$\mathrm{TRBAC_M}$ allows us to use the test system given in Fig. 9 for conformance testing of the given ACUT. This test system has been first proposed in [15] (there is a minor variation in our test system as the state queries are used to get the current state information from the ACUT). The purpose of the Test-Controller is to control the execution of all the test sequences of the conformance test suite. It sends inputs to the ACUT and *Set* events to the Clock-Handler at the specified times and receives the outputs from the ACUT and *Exp* events from the Clock-Handler. After each input or output event, the Test-Controller queries the ACUT to determine the current state. A test sequence is considered passed if all the outputs from the ACUT match the corresponding *Exp* event and the state information agrees with the one required by the test sequence. An ACUT is thus considered conforming if all the tests pass.

Each path in Tr represents a unique test sequence. For a given path p_t in Tr, the test sequence is constructed by associating all the edges $e \in p_t$ with monotonically increasing time stamps. Constraints are imposed on the time stamps by virtue of the semantics of se-$\mathrm{TRBAC_M}$. The time stamp associated with an input action indicates the time at which Test-Controller should generate the corresponding input for the ACUT and the Clock-Handler. An ACUT will pass the subject sequence if outputs are generated by the ACUT and the Clock-Handler at times corresponding to the time stamps of output actions.

To illustrate the semantics of a test sequence and the procedure used for determining the feasible value of time stamps so that the sequence can be executed by the test system on the ACUT, we consider as an example the following sequence obtained from the Tr of Fig. 8:

$$\begin{aligned} TS_1 = {} & q_0, (?a, t_1)q_0, (?b, t_2)q_0, (?c, t_3)q_1, (?b, t_4)q_1, \\ & (?e, t_5)q_1, (?f, t_6)q_3, (?b, t_7)q_3, (?e, t_8)q_3, (!d, t_9)q_4, \\ & (?a, t_{10})q_4, (?b, t_{11})q_4, (!g, t_{12})q_5, (?a, t_{13})q_5, (?b, t_{14})q_5 \end{aligned}$$

with the temporal constraints $t_1 = 0, t_{i+1} > t_i, t_9 - t_3 = 3$, and $t_{12} - t_6 = 1$.

TS_1 corresponds to the execution sequence of ACUT where the u_1r_1 activation is preempted by the deassignment output. The constraints $t_9 - t_3 = 3$ and $t_{12} - t_6 = 1$ represent the time difference between the matching *Set* and *Exp* events in this sequence. The feasible value of time stamps should satisfy the required temporal constraints, which can be represented as: $dt_2 = t_2, dt_i > 0, 2 \leq i \leq 12, dt_4 + dt_5 + dt_6 + dt_7 + dt_8 + dt_9 = 3$, and $dt_7 + dt_8 + dt_9 + dt_{10} + dt_{11} + dt_{12} = 1$, where $dt_i = t_i - t_{i-1}$.

This problem can be treated as a linear program by considering that the objective function $\sum_i dt_i$ is minimized subject to the given constraints. As we are dealing with dense time semantics, the obtained solution can have very minute resolution (the smallest value among all the dt_is); thus, the execution of the given sequence might not be practically possible. We overcome this problem by specifying the minimum resolution and formulating the feasible time stamp determination problem as an Integer Program (IP) [27]. The integer program corresponding to the above problem will be: $\min \sum_i dt_i$, subject to $dt_i/k = c_i, 2 \leq i \leq 12, dt_4 + dt_5 + dt_6 + dt_7 + dt_8 + dt_9 = 3, dt_7 + dt_8 + dt_9 + dt_{10} + dt_{11} + dt_{12} = 1$, and $c_i \in Z^+$, where k specifies the minimum resolution. Solving the IP for $k = 0.1$, the *test sequence* would be:

$$q_0, (?a, 0)q_0, (?b, 0.1)q_0, (?c, 0.2)q_1, (?b, 2.3)q_1, (?e, 2.4)q_1, (?f, 2.5)q_3, (?b, 3)q_3, (?e, 3.1)q_3, (!d, 3.2)q_4, (?a, 3.3)q_4, (?b, 3.4)q_4, (!g, 3.5)q_5, (?a, 3.6)q_5, (?b, 3.7)q_5.$$

When no feasible solution exists for a specific value of k, we can continue reducing the value of k until the time a solution is obtained. The CTS is thus obtained by determining the feasible time stamps for all the test sequences, where as mentioned above, each sequence corresponds to a unique path in the Tr.

7.4 Relation between TRBAC, TIOA, and se-FSA Fault Models

The TIOA-based fault model considered in [10] comprises two types of faults: timing faults and "action (output) and transfer." We considered an extended TIOA fault model with missing location and extra location faults, not considered in [10] because of the test hypothesis considered there. The action, transfer, missing, and extra location faults in TIOA are similar to the output (operation), transfer, missing, and extra state faults in finite-state machines [9].

There could be three types of timing faults in an ACUT′ [10]: 1) reset of a clock fault; 2) time constraint restriction fault; and 3) time constraint widening fault. An ACUT′ would have a clock reset fault if it does not reset the expected clocks or resets wrong clocks not stated by the specification. We consider the other two types of faults specifically with respect to our TIOA TRBAC_M model in which guards only restrict the timings of output transitions through equality constraints on the clock values. An ACUT′ would have a time constraint restriction fault (time constraint widening fault) if a constraint $x = t'$ is replaced by $x = t$ such that $t' < t (t' > t)$.

The relation between the TRBAC faults, described in Section 5, and the TIOA faults in the TRBAC_M model is illustrated in Table 2. As all the TRBAC faults can be associated with some fault type in the TRBAC_M model, a CTS capable of detecting all TRBAC_M faults would automatically guarantee complete fault coverage for TRBAC faults.

TABLE 2
Relation between TRBAC and TIOA Faults

TRBAC Faults	TIOA Faults
UR1, UA1, PR1	Transfer, Missing location, Output
UR2, UA2, PR2	Extra location, Output, Transfer
Hierarchical enforcement	Transfer, Output
Duration restriction	Time constraint restriction, clock reset
Duration widening	Time constraint widening, clock reset

The se-FSA-based fault model considered in [15] consists of only the output and transfer faults. We extend this fault model through inclusion of missing state and extra state faults. The output, transfer, missing location, and extra location faults in the TRBAC_M model have similar representation in the se-TRBAC_M. The time constraint restriction and widening faults are represented in the form of combination of output and transfer faults. However, a clock reset fault does not have any direct correspondence with the faults in the se-TRBAC_M model as the se-FSA model always considers pairs of *Set* and related *Exp* events, and thus, a missing or extra *Set* event carries no semantics in the se-TRBAC_M model.

7.5 Fault Coverage of CTS

CTS is generated by applying the W-method on se-TRBAC_M, where the W-method provides complete fault coverage for output, transfer, missing state, and extra state faults under the assumption that the number of states in the ACUT is accurately estimated [9]. In Section 7.2, we highlighted the issue of detecting such transfer, missing state, and extra state faults which lead to incorrect states with same location as of correct states but a differing range of clock variables. Note that states $q \in Q(S_E)$ of an se-FSA contain information corresponding to both locations of the source TIOA and the range of clock variables [16], referred here as ΔC. Consider that ACUT is the correct implementation corresponding to se-$\text{TRBAC}_\text{M} = S_E$, and there is a faulty ACUT′ which implements $S_{E'}$, where S_E and $S_{E'}$ differ only in $\delta(S_E)$ in case of a transfer fault in the ACUT′ and in both $Q(S_E)$ and/or $\delta(S_E)$ and $O(S_E)$ in case of a extra or missing state fault in the ACUT′. Consider the execution of ACUT′ against test sequence $TS \in \text{CTS}$ where transition $\tau = (q_i, (i, t)q) \in TS$ corresponds to fault $f = (q_i, (i', t)q'), q' \neq q$ in ACUT′ such that q and q' only differ in ΔC. Consider that TS corresponds to the path p_t in Tr.

Note that ΔC would only vary in q' from q if i' and i differ in Set or Exp or both events. If i and i' differ in Set events, which corresponds to clock reset fault, then as discussed next, such faults would be detected by the CTS as output faults. If the difference between the two is only in Exp events, then we separately consider the cases of missing, extra, or modified Exp events in i' in relation with the semantics of se-FSA. A missing Exp event would only occur if there is a missing clock reset fault earlier in the

path p_t before the edge corresponding to τ. An extra Exp event would similarly correspond to incorrect clock reset. A modified Exp would occur because of either a missing or incorrect clock reset fault or combination of two faults and as discussed later, the clock reset faults would be detected as output faults. From this discussion, it is simple to observe that if i' differs from i in both Set and Exp events, then this would also correspond to clock reset faults.

From the above discussion, we infer that if an ACUT′ has transfer, missing state, and extra state faults that lead to incorrect states with same location as of correct states but differing range of clock variables, then such faults would always occur in combination with output faults which would then be always detected by at least one element in CTS. Thus, if an ACUT′ passes the CTS, then it would be free of all the output, transfer, missing, and extra state faults. The correlation between TIOA and se-FSA faults, established in Section 7.4, implies that CTS would be able to detect all the output, transfer, missing and extra location, and time constraint restriction and widening faults in the TRBAC_M. We claim that the clock reset faults would also be detected by the CTS.

Next, consider a missing clock reset fault in the ACUT′. Note that the CTS includes at least one such test sequence which contains both the *Set* event associated with the missing reset and an output event containing the corresponding *Exp* event. As this sequence is executed against the faulty ACUT′, the time of occurrence of the output event and the *Exp* event would not be the same, and thus, the missing clock reset fault would be detected by at least one element of CTS as an output fault. By using a similar approach, it can be easily shown that the reset of incorrect clock fault would also be detected by the CTS. It is important to note that an incorrect clock reset fault would only alter the semantics of the TIOA model, in case if the corresponding clock is used in some guard subsequently (before its correct reset) in the semantic graph of TIOA.

Proposition 7.1. *CTS detects all transfer, output, missing and extra location, and timing faults in the $TRBAC_\text{M}$, and hence, it must detect all TRBAC faults in an ACUT′ given that there are no faults in the ACUT′ because of user/system-administrator-initiated deassignment/deactivation requests.*

The proof of Proposition 7.1 is based on the fact that CTS is able to detect all the faults in se-TRBAC_M, the correlation between fault models established in Section 7.4, and the correctness of TRBAC_M established by Theorem 6.1.

To illustrate the fault coverage of CTS, consider that corresponding to the ACUT which correctly enforces simplified P (Section 7.1) of Example 1, there are two faulty ACUTs: ACUT′ and ACUT″ which, respectively, enforce policies $TRBAC_{P'}$ and $TRBAC_{P''}$. $TRBAC_{P'} = (U, R, Pr, \ldots, S_s, D_s, \Re^1)$ and $TRBAC_{P''} = (U, R, Pr, \ldots, S_s, D_s, \Re^2)$, where $\Re^1$ differs from $\Re$ in only γ_{urSSoD} such that $\forall(u,r) \in U \times R\ \gamma_{urSSoD}(u,r) = 1$. $\Re^2$ differs from $\Re$ in γ_1 in enforcing the duration constraint on $AS(u,r,t)$ by increasing the duration to $t+1$ in the $TRBAC_{P''}$.

Table 3 records the results of executing ACUT and the two faulty ACUTs against TS_1. The various notations used in Table 3 correspond to: $I_s =$ initial state, $N_s =$ next state, (i,t) = (input, time), and $(o,t) =$ (output, time). The error in $TRBAC_{P'}$ would lead to a UR2 fault as despite the existence of static SoD constraint on simultaneous u_1 assignment to r_1 and r_2, the concurrent u_1r_1 and u_1r_2 assignments would exist in the ACUT′. The error in $TRBAC_{P''}$ would cause a duration widening fault as the duration of u_1r_1 assignment would be inappropriately extended to $4tu$s, against $3tu$s specified by P. As indicated by Table 3, the UR2 fault in $TRBAC_{P'}$ and the duration widening fault in $TRBAC_{P''}$ would be detected by TS_1 as extra state and output/transfer faults, respectively.

TABLE 3
Comparison of TS_1 Execution on Conforming and Faulty ACUTs

TS_1	ACUT	ACUT′	ACUT″
I_s	q_0	q_0	q_0
i,t	$?a,0$	$?a,0$	$?a,0$
N_s	q_0	q_0	q_0
i,t	$?b,0.1$	$?b,0.1$	$?b,0.1$
N_s	q_0	q_0	q_0
i,t	$?c,0.2$	$?c,0.2$	$?c,0.2$
N_s	q_1	q_1	q_1
i,t	$?b,2.3$	$?b,2.3$	$?b,2.3$
N_s	q_1	q_1	q_1
i,t	$?e,2.4$	$?e,2.4$	$?e,2.4$
N_s	q_1	q_1'	q_1
i,t	$?f,2.5$	q_1' differs	$?f,2.5$
N_s	q_3	from q_1 in	q_3
i,t	$?b,3$	the value of	$?b,3$
N_s	q_3	UR_{assign}	q_3
i,t	$?e,3.1$	(u_1,r_2) which is	$?e,3.1$
N_s	q_3	1 in the	q_3
o,t	$!d,3.2$	former and 0	$\mathbf{E}xp(x_1),3.2$
N_s	q_4	in the later	$\mathbf{q}_3$

8 Heuristics for CTS Reduction

Though promising, the test generation approach based on construction of CTS from TRBAC_M presented in Section 7.3 can be expensive, and thus, impractical in terms of the size of the model required to capture the ACUT behavior and the size of the corresponding CTS. We propose two heuristics, labeled as HT1 and HT2, to reduce the size of the model and the corresponding CTS.

8.1 Heuristic HT1

This heuristic considers a reduced TRBAC_M, referred to as $\text{TRBAC}_{\text{M}'}$. The size of $\text{TRBAC}_{\text{M}'} = \text{UR}_{\text{M}'} \parallel \text{PR}_{\text{M}'}$ is reduced by considering fewer number of URs and PRs, respectively, in the construction of $\text{UR}_{\text{M}'}$ and $\text{PR}_{\text{M}'}$ as compared to UR_M and PR_M. For $\text{UR}_{\text{M}'}$, the number of URs is reduced by considering URs corresponding to only those user-role pairs for which explicit assignment is provided by P, i.e., $\forall(u,r) \in D_1$; hence, $k = |D_1|$. Thus, in case of Example 1, although the total number of possible user-role assignments is four, i.e., assignments consequent to u_1r_1, u_1r_2, u_2r_1, and u_2r_2 pairs; however, only three URs are considered in constructing $\text{UR}_{\text{M}'}$ because u_2r_1 assignment is not explicitly stated by P.

By using the proposed strategy for reducing the number of URs, the size of the resultant $\text{UR}_{\text{M}'}$ can be significantly trimmed. However, this trimming is at the expense of reduced fault detection effectiveness of CTS. Specifically, $\text{UR}_{\text{M}'}$ does not encapsulate sufficient information which can reveal all the UR and UA faults in the ACUT′ (even under the assumption that the events corresponding to user-role

assignments/activations and permission-role assignments can be considered as independent). To lessen the impact of this shortcoming, in addition to the tests generated from $\mathrm{TRBAC}_{\mathrm{M}'}$, we suggest separate validation of all such user-role assignments and activations not captured by $\mathrm{UR}_{\mathrm{M}'}$.

The separate validation is performed by verifying that for the given user-role pairs, corresponding to the application of inputs AS and AC, the ACUT' response matches the one permitted by $TRBAC_P$. Note that this validation does not guarantee the absence of all the UR and UA faults in the ACUT' as there could be faults that are only depicted during a specific sequence of events. As an example consider a UR2 fault that leads to u_2r_1 assignment in the ACUT' corresponding to Example 1. If this fault is only visible after the occurrence of a u_2r_2 activation, i.e., when $UR_{active}(u_2, r_2) = 1$, then it cannot be revealed by such validation.

Similarly, the reduction in the size of $\mathrm{PR}_{\mathrm{M}'}$ is achieved by constructing the PR_{b}s for only those permission-role pairs for which explicit assignment is specified by the $TRBAC_P$, i.e., $\forall(p, r) \in D_3$; hence, $j = |D_3|$. This will likely lead to reduced fault detection effectiveness of the CTS. Specifically, $\mathrm{PR}_{\mathrm{M}'}$ does not capture enough information to reveal all the PR faults in an ACUT' (even under the assumption that the events corresponding to user-role assignments/activations and permission-role assignments can be considered as independent). Therefore, we again suggest separate validation of all such permission-role assignments which are not captured by $\mathrm{PR}_{\mathrm{M}'}$. The validation of all such permission-role pairs is performed by applying corresponding AP inputs to the ACUT' and comparing its response with the one specified by the $TRBAC_P$. As before, such validation does not guarantee the absence of all the PR faults in the ACUT' as there could be such faults which are only depicted during a specific sequence of inputs.

As an example, consider the $\mathrm{PR}_{\mathrm{M}'}$ illustrated in Fig. 6. Consider a PR2 fault in the ACUT' that leads to p_2 assignment to r_2 only after the occurrence of a p_2r_1 assignment, i.e., when $PR_{assign}(p_2, r_1) = 1$ becomes true. This fault cannot be revealed by the separate validation.

8.2 Heuristic HT2

In this heuristic, the size of the CTS is reduced by generating it independently from the UR_{M} and PR_{M} models. As UR_{M} and PR_{M} models do not capture the complete ACUT behavior specified by P; therefore, unless it is possible to assume that all the events in UR_{M} and PR_{M} are independent, complete fault coverage cannot be guaranteed. Note that by abstracting the details captured by a location in the TIOA-based TRBAC model, various other heuristics can be designed such as constructing separate TIOA models for each user and each role.

9 Related Work

TIOA is proposed by Alur and others [3], [5]. Several researchers have proposed test generation for real-time systems as in [8], [10], [15], [18]. Although there exist other formalisms such as timed Petri nets, timed process algebras, and real-time logics [5] for specifying real-time systems, we selected TIOA as it leverages the significant amount of research on test generation from timed automata. The proposed test generation procedure (Section 7) is inspired by the work of Khoumsi [15]. The se-FSA technique provides good fault coverage without the disadvantage of significant loss in scalability. Although the first step, i.e., se-FSA transformation of $\mathrm{TRBAC}_{\mathrm{M}}$ given in Section 5 is similar to Khoumsi's approach, subsequent steps differ considerably. Our se-FSA transformation results in a deterministic FSA. Also, the ability to directly monitor the states considerably simplifies the CTS generation. We have also studied the problem of making the tests executable by determining the time stamps at which inputs should be generated and at which corresponding outputs should occur.

Timed-Wp method [10] provides complete fault coverage of TIOA faults but is not scalable. The exponential complexity of timed-Wp is primarily due to the construction method used. However, for most TIOAs, the state space of se-FSAs does not increase with the magnitude of constants used in timing constraints (it only increases with the number of clocks) [16]. For comparison, the region graph of $\mathrm{UR}_{\mathrm{b}}(u_1, r_1)$ contains 47 states as compared to only 8 states in its se-FSA. The size of the NTFSM corresponding to region graph of $\mathrm{UR}_{\mathrm{b}}(u_1, r_1)$, from which tests would be generated, will be even larger by virtue of sampling. Another issue with the general applicability of the Timed-Wp method is that the fault coverage is only guaranteed for a specific Implementation Under Test (IUT) architecture by assuming that clock resets are observable.

The approaches in [18], [19] are based on symbolic clustering of states into partitions coarser than regions, and thus, are better scalable as compared to region-graph-based test generation techniques. With some minor variations, both approaches consider conformance between the specification and the IUT as a timed input-output conformance (tioco) relation, i.e., for all the traces of the specification, the IUT always produces outputs within the given temporal bounds. These approaches do not consider any fault model or provide any guarantees of fault coverage. The *digital-clock* test generation of [17] and [18] is similar to our IP-based approach used in CTS construction in Section 7.3 in terms of the semantics of the generated test sequences. Table V in Appendix VI summarizes the comparison between scalability and fault detection effectiveness of various timed automaton test generation approaches and the proposed CTS.

10 Summary and Discussion

A technique for behavior modeling of TRBAC systems and a conformance testing procedure for TRBAC ACUTs is proposed. The proposed procedure provides complete fault coverage with respect to a proposed TRBAC fault model. The fault model is obtained by following the mutation-based approaches described in [23]. The complete fault coverage of the generated CTS is by virtue of the correctness of the TIOA-based behavior modeling technique presented in Section 6, and the correlation between TRBAC, TIOA, and se-FSA faults established in Section 7.4.

The proposed conformance testing technique is based on a transformation of $\mathrm{TRBAC}_{\mathrm{M}}$ to se-$\mathrm{TRBAC}_{\mathrm{M}}$, and then, using the W-method to generate the testing tree (Tr). CTS is then constructed from the Tr by using an IP-based approach that ensures that the test sequences satisfy the temporal constraints by only considering sending of inputs and monitoring of outputs at some integral multiple of minimum resolution k. Finally, we use a specific test system architecture to execute the CTS against the ACUT and compare the results so as to validate the ACUT conformance with respect to $TRBAC_P$. In Section 8, we show how to reduce the size of

the CTS through two heuristics based on state abstraction. The decision on whether to use the complete CTS or a smaller test suite can be based on the extent of resources available for the testing process and the desired level of fault detection effectiveness.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their constructive comments and suggestions. This research has been supported by the US National Science Foundation under NSF Grants IIS-0209111 and IIS-0242419, and a grant from the Software Engineering Research Center.

REFERENCES

[1] T. Ahmed and A.R. Tripathi, "Static Verification of Security Requirements in Role Based CSCW Systems," *Proc. Eighth ACM Symp. Access Control Models and Technologies (SACMAT '03),* pp. 196-203, 2003.
[2] G.-J. Ahn and R. Sandhu, "Role-Based Authorization Constraints Specification," *ACM Trans. Information System Security,* vol. 3, no. 4, pp. 207-226, 2000.
[3] R. Alur and D.L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science,* vol. 126, no. 2, pp. 183-235, 1994.
[4] S. Barker and P.J. Stuckey, "Flexible Access Control Policy Specification with Constraint Logic Programming," *ACM Trans. Information and System Security,* vol. 6, no. 4, pp. 501-546, 2003.
[5] B. Berthomieu and M. Diaz, "Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 17, no. 3, pp. 259-273, Mar.-Apr. 1991.
[6] E. Bertino, P.A. Bonatti, and E. Ferrari, "Trbac: A Temporal Role-Based Access Control Model," *ACM Trans. Information and System Security,* vol. 4, no. 3, pp. 191-233, 2001.
[7] R.V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley Longman Publishing Co., Inc., 1999.
[8] R. Cardell-Oliver, "Conformance Tests for Real-Time Systems with Timed Automata Specifications," *Formal Aspects of Computing,* vol. 12, no. 5, pp. 350-371, 2000.
[9] T.S. Chow, "Testing Software Design Modelled by Finite State Machines," *IEEE Trans. Software Eng.,* vol. 4, no. 3, pp. 178-187, May-June 1978.
[10] A. En-Nouaary, R. Dssouli, and F. Khendek, "Timed wp-Method: Testing Real-Time Systems," *IEEE Trans. Software Eng.,* vol. 28, no. 11, pp. 1023-1038, Nov.-Dec. 2002.
[11] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Trans. Information and System Security,* vol. 4, no. 3, pp. 224-274, 2001.
[12] A. Gal and V. Atluri, "An Authorization Model for Temporal Data," *Proc. ACM Conf. Computer and Comm. Security,* pp. 144-153, 2000.
[13] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Information and Computation,* vol. 111, no. 2, pp. 193-244, 1994.
[14] J.B.D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A Generalized Temporal Role-Based Access Control Model," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 1, pp. 4-23, Jan. 2005.
[15] A. Khoumsi, "A Method for Testing the Conformance of Real Time Systems," *Proc. Seventh Int'l Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT),* pp. 331-354, 2002.
[16] A. Khoumsi and L. Ouedraogo, "A New Method for Transforming Timed Automata," *Electronic Notes in Theoretical Computer Science,* vol. 130, pp. 101-128, 2005.
[17] M. Krichen and S. Tripakis, "Black-Box Conformance Testing for Real-Time Systems," *Proc. 11th Int'l SPIN Workshop Model Checking Software,* pp. 109-126, 2004.
[18] M. Krichen and S. Tripakis, "An Expressive and Implementable Formal Framework for Testing Real-Time Systems," *Proc. 17th IFIP TC6/WG 6.1 Int'l Conf. Testing of Comm. Systems (TestCom '05),* pp. 209-225, 2005.
[19] K.G. Larsen, M. Mikucionis, and B. Nielsen, "Online Testing of Real-Time Systems Using UPPAAL," *Proc. Int'l Workshop Formal Approaches to Testing of Software (FATES '04),* pp. 79-94, 2004.
[20] E.C. Lupu and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management," *IEEE Trans. Software Eng.,* vol. 25, no. 6, pp. 852-869, Nov.-Dec. 1999.
[21] F. Malamateniou, G. Vassilacopoulos, and P. Tsanakas, "A Workflow-Based Approach to Virtual Patient Record Security," *IEEE Trans. Information Technology in Biomedicine,* vol. 2, no. 3, pp. 139-145, Sept. 1998.
[22] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur, " Scalable and Effective Test Generation for Role-Based Access Control Systems," *IEEE Trans. Software Engineering* vol. 35, no. 5, pp. 654-668, Sept. 2009.
[23] A. Petrenko, G.V. Bochmann, and M. Yao, "On Fault Coverage of Tests for Finite State Specifications," *Computer Networks and ISDN Systems,* vol. 29, no. 1, pp. 81-106, 1996.
[24] R. Sandhu, "Role Activation Hierarchies," *Proc. Third ACM Workshop Role-Based Access Control (RBAC '98),* pp. 33-40, 1998.
[25] J. Springintveld, F.W. Vaandrager, and P.R. D'Argenio, "Testing Timed Automata," *Theoretical Computer Science,* vol. 254, nos. 1/2, pp. 225-257, 2001.
[26] S. Tripakis1 and S. Yovine, "Analysis of Timed Systems Using Time-Abstracting Bisimulations," *Formal Methods in System Design,* vol. 18, no. 1, pp. 25-68, 2001.
[27] L.A. Wolsey, *Integer Programming.* John Wiley, 1998.

Ammar Masood received the PhD degree from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana. He is currently working as an assistant professor in the Department of Avionics Engineering, Institute of Avionics and Aeronautics, Air University, Islamabad, Pakistan. His research interests include information system security, software security testing and verification, access control systems, multimedia systems, and networking.

Arif Ghafoor is currently a professor in the School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, and is the director of Distributed Multimedia Systems Laboratory. He has been actively engaged in research areas related to multimedia information systems, database security, and parallel and distributed computing. He has published numerous papers in these areas. He has served on the editorial boards of various journals including the *ACM/Springer Multimedia Systems Journal*, the *Journal of Parallel and Distributed Databases*, and the *International Journal on Computer Networks*. He has served as a guest/coguest editor for various special issues of numerous journals including the *ACM/Springer Multimedia Systems Journal*, the *Journal of Parallel and Distributed Computing*, the *International Journal on Multimedia Tools and Applications*, the *IEEE Journal on the Selected Areas in Communications*, and the *IEEE Transactions on Knowledge and Data Engineering*. He has coedited a book entitled *Multimedia Document Systems in Perspectives* and has coauthored a book entitled *Semantic Models for Multimedia Database Searching and Browsing* (Kluwer Academic Publishers, 2000). He is a fellow of the IEEE. He has received the IEEE Computer Society 2000 Technical Achievement Award for his research contributions in the area of multimedia systems.

Aditya Mathur has research span in software testing, reliability, process control, and a study of the function of the human brain. His published work relates to investigations into the effectiveness of testing techniques and their applicability to the testing of sequential and distributed software, methods for the estimation of software reliability, and techniques and tools for managing a collection of Internet-enabled devices. His recent work is in software process control to formalize control of the software development process using methods from control theory. His research contributions are contained in more than 100 publications in international journals, conference proceedings, and magazines. He is the author of several textbooks including the recently published *Foundations of Software Testing*. He is a member of the IEEE.