Active Authorization Rules for Enforcing Role-Based Access Control and its Extensions*

Raman Adaikkalavan and Sharma Chakravarthy
Information Technology Laboratory and Department of Computer Science & Engineering
The University of Texas at Arlington
{adaikkal, sharma}@cse.uta.edu

Abstract

Dynamically monitoring the state changes of an underlying system, detecting and reacting to changes without delay are crucial for the success of any access control enforcement mechanism. With their inherent nature, active (Event-Condition-Action or ECA) rules are prospective candidates to carry out change detection and to provide access control. Current systems or models do not provide a flexible mechanism for enforcing Role-Based Access Control (RBAC) standard and its extensions in a seamless way, and do not adapt to policy or role structure changes in enterprises, which are indispensable to make RBAC usable in diverse domains.

In this paper we will show how On-When-Then-Else authorization rules (or enhanced ECA rules) are used for enforcing RBAC standard and its extensions such as generalized temporal RBAC, control flow dependency constraints, privacy-aware RBAC, and so forth in a seamless way. Furthermore, these rules also provide active security. Large enterprises have hundreds of roles, which requires thousands of rules for providing access control, and generating these rules manually is error-prone and a cognitive-burden for non-computer specialists. Thus, in this paper, we will discuss briefly how these authorization rules can be automatically (or semi-automatically) generated from high level specifications of enterprise access control policies. We will also discuss the implementation using Sentinel+, an active object oriented system.

1 Introduction

With the ever growing impact of computing systems on our daily activities, security and privacy have a greater role to play. Role-Based Access Control [18, 29], where object accesses are controlled by roles (or job functions) in an enterprise rather than a user or group, has proven to be a positive alternative to traditional access control mechanisms. RBAC does not provide a complete solution for all access control issues, but with its rich specification it has proven to be cost effective [26] by reducing the complexity in authorization management of data. Lately, RBAC has been standardized [21] and is being extended for handling various constraints such as temporal [22, 23], context-aware [25, 5], privacy-aware [19], control flow dependency [23], and so forth, so that it can support diverse domains in authorization management of data.

Even though lot of work has been carried out in the specification of RBAC and its extensions, there are still lots of work that need to be carried out in enforcing these specifications in a seamless way, which is essential for making RBAC better-suited for enterprises in diverse domains. Enterprises can formalize their access control (or security) policies using RBAC or its extensions that provide additional constraints. Enterprises in different domains have different requirements; for example, health care domain requires extensive temporal [22] and context-aware constraints (e.g., emergency room, intensive care).

Current systems and models do not provide a generalized approach for enforcing RBAC and its extensions in a seamless manner. For example, some systems just support separation of duty (SoD) relations, but without role hierarchies. Similarly they do not adapt to policy or role structure *changes* in enterprises, which is indispensable for making RBAC usable. For example, when an enterprise wants to change its working hours of a role, then the low level semantic descriptors[1] have to be modified. In the current systems it is a burden on the administrator to modify and maintain these low level semantic descriptions such as authorization rules, manually.

Taking timely actions based on the state changes of the underlying system over a period of time and alerting the administrator regarding the malicious activities will comple-

*This work was supported, in part, by NSF grants IIS-0326505 and IIS-0123730.

[1] In this paper we refer to authorization rules, java classes, and other mechanisms that are used to enforce the access control as low level semantic descriptors.

ment the access control system. For example, all the constraints that are satisfied by an user when activating a role should hold TRUE until the role is deactivated. When any one of the constraints become FALSE before deactivation, then that role should be deactivated. Prevention of malicious activities in the system plays a major role while providing security. Enterprises require the detection and prevention of malicious activities from causing damage, without human intervention. Furthermore it will ameliorate the security of the underlying system so that enterprises can be more secure. For example, when access requests by unauthorized roles for some files are more than a certain number of times within a duration, an internal security alert is triggered and some critical authorization rules are disabled[2] and the administrators are alerted.

Existing systems (or models) [9, 30, 32, 2, 31, 13, 17, 14] enforcing RBAC are custom-implemented, domain-specific and are confined to particular form of constraints. All these systems neither enforce complete RBAC standard nor provide a generalized approach for enforcing it. Furthermore, they do not provide an approach for enforcing RBAC extensions in a seamless manner. Thus, systems (or models) that need to enforce RBAC in a generalized manner should be able to provide uniform and transparent handling of RBAC standard and its extensions, adapt to policy and role structure changes in an enterprise, and support high level specification of enterprise access control policies.

In this paper we will address the following; *i)* introduce Event-based active authorization rules or OWTE rules, *ii)* synthesis of active authorization rules for access control enforcement, *iii)* discuss how RBAC standard and its extensions are enforced in a seamless way, *iv)* how active security is provided, and *v)* show how initial set of rules are created using Sentinel+ based on an enterprise access control policy and show how the rules are regenerated when there is a change in the access control policies. Sentinel+ is an enhanced version of Sentinel [15, 12], an active object-oriented system.

The rest of the paper is as follows. Brief introduction for RBAC is given in Section 2. Event-Based active authorization rules are introduced in Section 3. Synthesis of active authorization rules for access control enforcement, rules illustrating the enforcement of RBAC, its extensions, and active security are provided in Section 4. Implementation, generation and regeneration of rules are briefly explained in Section 5. Related work is presented in Section 6. Conclusions are provided in Section 7.

2 Role-Based Access Control

NIST RBAC Standard [21] is defined in terms of four model components and their restricted combinations: 1) *Core RBAC:* defines relationships between three basic elements (i.e., users, roles, permissions). Permissions consist of objects and associated operations that can be performed on those objects. 2) *Hierarchical RBAC:* defines hierarchies between roles. "A hierarchy is mathematically a partial order defining a seniority relation between roles, whereby senior roles acquire the permissions of their juniors, and junior roles acquire the user membership of their seniors" [21]. 3) *Static Separation of Duty (SoD) Relations:* used to enforce conflicts of interest policies which may arise as a result of user gaining permissions to conflicting roles. Static SoD relations prevent these conflicts between roles by placing constraints on the assignment of users to roles. and 4) *Dynamic SoD Relations:* these are similar to the static SoD that limits user permissions, but they differ by the context in which the constraints are placed. A user can be assigned to $\mathcal{M}$ (i.e., two or more) mutually exclusive roles, but cannot be active in $\mathcal{N}$ or more mutually exclusive roles at the the same time, where $\mathcal{N} \geq 2$ and $\mathcal{N} \leq \mathcal{M}$.

[2]Actions are predefined by the security administrators.

3 Event-Based Active Authorization Rules

We will introduce On-When-Then-Else or OWTE authorization rules, which are Event-Condition-Action or ECA rules enhanced with additional functionalities such as alternative actions and enhanced operator semantics to support authorization management of data. (*Note: In this paper we will use active authorization rules, active rules, authorization rules, and OWTE rules interchangeably.*)

Active authorization rules consist of five components and they are 1) Rule name (or $\mathcal{R}_{name}$), 2) "O" an event (or an occurrence of interest) $\mathcal{E}_i$ that triggers a set of rules, 3) "W" checks the conditions $< \mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_n >$ when an associated event is triggered, 4) "T" triggers a set of actions $< \mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_n >$ when the conditions evaluate to TRUE, and 5) "E" triggers a set of alternative actions $< \mathcal{AA}_1, \mathcal{AA}_2, \ldots \mathcal{AA}_n >$ when the conditions evaluate to FALSE. An event occurrence can trigger rules that can be in the form of multiple rules, nested/cascaded rules, prioritized rules, and causality rules. OWTE rules are specified as shown below

```
RULE [  $\mathcal{R}_{name}$
        ON     Event $< \mathcal{E}_i >$
        WHEN   $< \mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_n >$
        THEN   $< \mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_n >$
        ELSE   $< \mathcal{AA}_1, \mathcal{AA}_2, \ldots \mathcal{AA}_n >$  ]
```

Simple Events: An event is an occurrence of interest in an application or a system. All the events that are predefined in the underlying system (i.e., domain-specific) are known as primitive or simple events. *File operations* (i.e., opening, closing, etc.) in operating systems, *method execution* by objects in object oriented systems, *data manipulations* such as insert, delete and update in relational database management systems, *system clock* of the underlying system (i.e., absolute or relative temporal events), *external* events (i.e., based on the data from sensors), and so forth are all simple events. For instance, below shown is an event $\mathcal{E}_i$ that is detected when a function $\mathcal{F}$ is invoked by an object $\mathcal{U}$. Parameters ($< \mathcal{PA}_1, \mathcal{PA}_2, \ldots \mathcal{PA}_n >$) are used by OWTE rules for checking conditions and performing actions.

Event $\mathcal{E}_i = \mathcal{U} \rightarrow \mathcal{F}(< \mathcal{PA}_1, \mathcal{PA}_2, \ldots \mathcal{PA}_n >)$

Some of the above mentioned events are used to enforce various functionalities of RBAC. For example, when *a user moves from one location to another*, external events can trigger some rules that can "activate/deactivate" roles.

Conditions: Multiple conditions i.e., $< \mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_n >$ can be associated with an event. These conditions are evaluated when an event occurs. For example, when an user tries to *open* a protected file in a pervasive computing domain, the system can check whether the network is *secure* or *insecure* and can take decisions accordingly.

Actions and Alternative Actions: Once events are detected and all the associated conditions are evaluated to TRUE, predefined system critical actions (i.e., $< \mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_n >$) are performed. For example, when an internal security is triggered, the system has to take the following actions; *i)* generate reports and alert administrators, *ii)* deactivate a set of roles, *iii)* demote certain roles' permissions, and *iv)* block access requests or impose certain access restrictions. On the other hand, current event processing models do not handle when the conditions are evaluated to FALSE. In OWTE rules, alternative actions $< \mathcal{AA}_1, \mathcal{AA}_2, \ldots \mathcal{AA}_n >$ are triggered when the condition evaluation returns FALSE. Alternative actions are critical in authorization management of data. For example, when the user is in the insecure network then the protected file access should be *denied*.

Rule 1 (Rule with a Simple Event) *Create a rule that checks for permissions when user Bob tries to open a file "patient.dat" using the command* [3] *"vi (patient.dat)".*

```
EVENT    $E_1 = Bob \rightarrow vi(patient.dat)$

RULE [   $\mathcal{S}_1$
         ON      $E_1$
         WHEN  if (checkaccess(Bob, patient.dat) is TRUE)
                       return TRUE; else return FALSE;
         THEN   $< allowopening\ patient.dat >$
         ELSE    raise error "insufficient privileges"   ]
```

When user Bob opens the file i.e., **O**, permissions are checked using "checkaccess" i.e., **W**; if Bob has the permission (i.e., if condition returns TRUE), **T**hen file is opened, **E**lse *error* is raised.

Complex Events: In addition to simple events, complex events are often required in many situations. Using complex or composite events additional constraints can be placed on event occurrences while providing access control. Complex events are composed of more than one simple or complex event using event operators [1, 11]. Some of the event operators are AND, OR, NOT, SEQUENCE, Periodic, Aperiodic, and PLUS. We have enhanced the event operators so that they can support access control. We have explained some of these operators below in the context of RBAC and its extensions even though all of them are critical in access control.

SEQUENCE (E_1, E_2): When two events[4] E_1 and E_2 occur (i.e., "O"), a SEQUENCE event is detected and the corresponding rules are triggered. With this event operator, E_1 should occur before E_2. The condition that *a user should be active in role A to activate role B* (i.e., prerequisite roles in RBAC) can be specified using this event operator.

OR (E_1, E_2): This event is detected when any one of the two events E_1 and E_2 occur and the corresponding rules are triggered.

PLUS (E_1, Δ): This event is a relative temporal event. A simple or composite event occurrence starts a PLUS event (i.e., at time "T"). After the specified time interval or duration "Δ" (i.e., at time "$T + \Delta$") the PLUS event is detected. For example, *a user can be deactivated from a role after a certain duration "Δ"* using this operator.

APERIODIC (E_1, E_2, E_3): This event is detected whenever event E_2 occurs between two other events E_1 and E_3. Event E_1 starts the Aperiodic event and E_3 terminates the same. Event occurrences of E_2 cannot detect an Aperiodic event before the occurrence of event E_1 or after E_3. Only when event E_2 occurs within E_1 and E_3, an Aperiodic event is detected and the corresponding rules are triggered. For instance, using Aperiodic *a role can be allowed to be enabled only between a transaction.*

PERIODIC (E_1, τ, E_3): This event is similar to Aperiodic event except that it is detected at a regular time interval "τ" between two other events E_1 and E_3. This event operator can be used to *periodically monitor the underlying system and generate reports.*

All the above mentioned operators are critical and are necessary for providing authorization management of data. Apart from the examples provided, all the operators are also used for supporting various other functionalities of RBAC. Even though there are other operators, we have explained only a few in the context of RBAC and its extensions. A rule involving event operator PLUS is shown below.

Rule 2 (Rule with Complex Event) *Create a rule for restricting user Bob from keeping the file "patient.dat" open for more than 2 hours (i.e., Δ). In other words, close the file forcefully after 2 hours.*

```
RULE [   $\mathcal{C}_1$
         ON       PLUS($E_1$, 2 hours)
         WHEN  TRUE
         THEN   $< Closefile >$   ]
```

[3]We have used vi(patient.dat) to indicate that the file is opened using vi editor. This is just for understanding and it is not the case always.

[4]Events are represented as E_i in all the operators and they can be both simple and complex.

In the above rule, PLUS event[5] is started when user Bob opens the file “patient.dat” using the “vi” editor (i.e., event E_1 from Rule 1). This event is detected when the duration Δ (i.e., 2 hours) is elapsed, and the file is closed forcefully.

4 Synthesis of Active Authorization Rules for Access Control Enforcement

In this section we show the mapping between the basic elements in RBAC and its extensions and the OWTE rule specification. In doing so, we are establishing OWTE rules as an enforcement mechanism for the realization of access control policies. In addition we will demonstrate the following; 1) rules enforcing RBAC, 2) rules enforcing RBAC extensions, and 3) rules supporting active security.

4.1 Entity Relationship Modeling

RBAC contains three basic element sets namely users (or $\mathcal{U}$), roles (or $\mathcal{R}$), and permissions (or $\mathcal{P}$). In addition to these basic elements, RBAC extensions have additional elements such as “purpose” and “object-policy” in privacy-aware RBAC [19] and so forth. $\mathcal{U}$ represents humans, user applications and so forth, $\mathcal{R}$ represents a job function in an enterprise, $\mathcal{P}$ represents the operations that can be carried out on objects by $\mathcal{R}$, and “purpose” represents business purposes[6]. All the basic elements are considered as entities as they represent something that has a separate existence or conceptual reality. All the users (i.e., humans, user agents, etc.) and roles (e.g., manager, cashier, etc.) in an enterprise are modeled as *entity instances* of the basic entities $\mathcal{U}$ and $\mathcal{R}$, respectively.

Entities $\mathcal{U}$ and $\mathcal{R}$ have M:N (i.e., many-to-many) relationship. Similarly, entities can be associated with other entities by the means of role-permission assignments, role hierarchies, purpose hierarchies, and so forth. Thus, associations between the entities represents their *relationships* forming an Entity Relationship [28] like model. In this paper we will consider only the entities $\mathcal{U}$ and $\mathcal{R}$ and their relationships while discussing our approach. On the other hand, constraints such as separation of duty relations, temporal, context-aware, active security, and others are placed on the relationships [7] so that only entity instances that satisfy the constraints are allowed to take part in the relationship. For example, constraints that are placed on user-role activation are checked when an instance of entity $\mathcal{U}$ needs to take part in the relationship with an instance of entity $\mathcal{R}$.

4.2 Mapping OWTE and RBAC Elements

Active authorization rules are triggered when an event (i.e., “O”) is detected. In RBAC, only instances of entity $\mathcal{U}$ can trigger simple events as they alone can request accesses, assignments, activations, and so forth. On the other hand, temporal events, external events such as locations from sensors, and so forth can trigger simple events with RBAC extensions and active security. For example, when an user (i.e., instance of $\mathcal{U}$) tries to activate a role, it will triggering an event (i.e., “O”). Similarly, complex events are always triggered by simple events as they compose more than one simple or complex event.

All the constraints that can be specified in RBAC and its extensions can be enforced by placing them appropriately either in the condition part (i.e., “W”) of the rule or by using complex event operators. For example, when a user tries to activate a role by triggering an event, conditions will check whether $\mathcal{U}$ has the permissions to be active in $\mathcal{R}$. On the other hand, constraints such as prerequisite roles, maximum number of active roles for an user, how long a user can be active in a role, etc. can be specified using complex events.

When an event is detected and all the constraints are satisfied, the user should be allowed to be perform the *requested* operation. When “W” returns FALSE alternative actions are taken and the user request is denied.

Thus, instances of entity $\mathcal{U}$, clock-events, etc. act as events, constraints are placed using complex events and conditions, and operations such as allowing/denying access requests, assignments, activations/deactivations, and so forth act as the actions and alternative actions.

4.3 Enforcement using Active Rules

In this section we will demonstrate the use of active rules for enforcing certain functionalities of RBAC and its extensions, and for providing active security. On the other hand, OWTE rules shown in this section are **not** created manually by administrators. Generation of authorization rules along with their implementation are discussed in Section 5.

All the active authorization rules that are generated form a *rule pool*. Three kinds of rules are available in the rule pool and they are *i)* administrative rules, *ii)* activity control rules, and *iii)* active security rules. Administrative rules are used with high level specification of access control policies, activity control rules are used to control the activities that can be performed by the instances of $\mathcal{U}$, and active security rules are used for monitoring the state changes and taking preventive measures. Rules are generated at different granularities within each classification. *Specialized* rules that are specific to an instance of $\mathcal{U}$ (e.g., Bob), *localized* rules are specific to a particular role and are created based on the role properties, and *globalized* rules are generalized and are not specific to any role.

Let us take three simple scenarios; 1) user Jane should be restricted to a maximum of five active roles at a time, 2) role Programmer can be activated only by five users at a time, and 3) user Jim needs to be assigned to a role. Au-

[5]In Rule 2, the PLUS event in the “ON” clause can be a named event as in Rule 1 i.e., “ON E_2”, where Event E_2 = PLUS($E_1, 2hours$).

[6]The purpose for which an operation is executed [19].

[7]these are similar to descriptive attributes [28] in ER model but for different purposes

thorization rule corresponding to scenario 1 is a *specialized* rule as it restricts a particular user Jane from being active in more than five roles. On the other hand, for scenario 2, the rule should be based on the role as there can be many users who can be active in a role. Thus, a *localized* rule that correspond to a particular role is created to limit the number of active users. On the contrary, for scenario 3, user Jim should be assigned to a role and it can be any role. This rule can be *globalized* so that it can control all the user-role assignments (i.e., same rule is invoked with different parameters). Rules corresponding to scenarios 1 and 2 are activity control rules whereas 3 is an administrative rule.

4.3.1 Rules Illustrating RBAC Enforcement

We will demonstrate the enforcement of core, role hierarchies, static SoD, and dynamic SoD with the following active authorization rules (due to lack of space we will only provide some sample rules that demonstrates the ability of OWTE rules).

Rule 3 (Add Active Role) *Assume that a user is assigned to role R1. In order to perform some operations that are allowed for R1 he has to activate R1. The rule activates R1 by adding R1 to the active role set of that user session.*

```
EVENT     $E_2 = user \rightarrow AddActiveRole$R1(sessionId)
```

When the user tries to activate role R1, the function "AddActiveRoleR1" is invoked with the users session identifier (or sessionId) as its parameter. This raises event E_2, which in turn triggers the rule that checks whether the user can be activated in role R1. Below shown are four rules that correspond to different role properties;

```
RULE [    $\mathcal{AAR}_1$
          ON     $E_2$
          WHEN  (user IN userL) && (sessionId IN sessionL) &&
                (sessionId IN checkUserSessions(user)) &&
                (R1 NOT IN checkSessionRoles(user)) &&
                (checkAssignedR1(user) IS TRUE)
          THEN  addSessionRoleR1(sessionId)
          ELSE   raise error "Access Denied Cannot Activate"   ]
```

Rule $\mathcal{AAR}_1$ is used when role R1 does not take part in any relationship (i.e., *core RBAC*) such as hierarchies and SoD relations. First it checks whether user is available in list userL[8], then it checks whether the sessionId exists in list sessionL and whether the session is owned by that user. Once verified, it checks whether the user is assigned to role R1 using the function "checkAssignedR1" as a user should be assigned in order to activate any role. It then checks whether the role R1 is not activated in that session using "checkSessionRoles". Once all the above conditions are verified, role R1 is activated in that user session by invoking the function "addSessionRoleR1" and adding it to the *active role set*[9].

[8]We assume that users lists, role lists, session lists that contain user, role and session information, respectively, are already available. In addition we also assume that other functions that are used in the rule are also available.

[9]The set containing all the active roles for an user.

```
RULE [    $\mathcal{AAR}_2$
          ON     $E_2$
          WHEN  (user IN userL) && (sessionId IN sessionL) &&
                (sessionId IN checkUserSessions(user)) &&
                (R1 NOT IN checkSessionRoles(user)) &&
                (checkAuthorizationR1(user) IS TRUE)
          THEN  addSessionRoleR1(sessionId)
          ELSE   raise error "Access Denied Cannot Activate"   ]
```

Rule $\mathcal{AAR}_2$ is used when role R1 takes part in *general role hierarchies*. All the conditions are same as in rule $\mathcal{AAR}_1$ except one condition that checks whether the user is authorized to that role using the function "checkAuthorizationR1" instead of "checkAssignedR1" . This is carried out as the user can activate role R1 if he is assigned to role R1 or to any of its senior role.

```
RULE [    $\mathcal{AAR}_3$
          ON     $E_2$
          WHEN  (user IN userL) && (sessionId IN sessionL) &&
                (sessionId IN checkUserSessions(user)) &&
                (R1 NOT IN checkSessionRoles(user)) &&
                (checkAssignedR1(user) IS TRUE) &&
                (checkDynamicSoDSet(user, R1) IS TRUE)
          THEN  addSessionRoleR1(sessionId)
          ELSE   raise error "Access Denied Cannot Activate"   ]
```

Rule $\mathcal{AAR}_3$ shown below is used when role R1 takes part in a *dynamic SoD relation without hierarchies*. This rule is similar to rule $\mathcal{AAR}_1$ but with additional conditions for checking whether the dynamic SoD constraints are satisfied. Function "checkDynamicSoDSet" checks whether the addition of role R1 to the old active role set of the user satisfies the mutual exclusive constraints discussed in Section 2.

```
RULE [    $\mathcal{AAR}_4$
          ON     $E_2$
          WHEN  (user IN userL) && (sessionId IN sessionL) &&
                (sessionId IN checkUserSessions(user)) &&
                (R1 NOT IN checkSessionRoles(user)) &&
                (checkAuthorizationR1(user) IS TRUE) &&
                (checkDynamicSoDSet(R1) IS TRUE)
          THEN  addSessionRoleR1(sessionId)
          ELSE   raise error "Access Denied Cannot Activate"   ]
```

Rule $\mathcal{AAR}_4$ is used when role R1 takes part in *dynamic SoD relation with hierarchies*. This rule is similar to $\mathcal{AAR}_2$ but with additional conditions for checking whether dynamic SoD constraints are satisfied, similar to $\mathcal{AAR}_3$.

Similar to all the above scenarios activating a role that has *static SoD relation* will use rule $\mathcal{AAR}_2$ if role R1 takes part in role hierarchies or rule $\mathcal{AAR}_1$ is used if it does not take part.

Rule 4 (Cardinality Constraints) *Restrict the number of users who can be active in a role at the same time. For instance, there is only one person in the role of a university president. Below shown rule allows only five users to be active in role R1 at a time. Similarly, the number of roles a user can be active at the same time can also be restricted.*

```
EVENT     $E_3 = addSessionRoleR1(sessionId)$
EVENT     $E_4 = removeSessionRoleR1(sessionId)$
```

```
RULE [   $\mathcal{CC}_1$
         ON     $E_3$
         WHEN  if (CardinalityR1(INCR) IS TRUE) return TRUE
                   else return TRUE
         THEN   perform action <add role R1 to session with sessionId>
         ELSE    raise error "Maximum Number of Roles Reached"   ]

RULE [   $\mathcal{CC}_2$
         ON     $E_4$
         WHEN  TRUE
         THEN   CardinalityR1(DECR)   ]
```

Cardinality constraint for the above mentioned scenario requires to restrict the *role activation* so that no more than 5 users are activated. When the user tries to activate role R1 it triggers any one rule from $\mathcal{AAR}_1 \ldots \mathcal{AAR}_4$ based on the access control policy and role property, since these rules are used to activate the role R1. When the user satisfies all the conditions then the function "addSessionRoleR1" is invoked in order to add the role R1 to *active role set* of that user session. This function raises event E_3 which in turn raises rule $\mathcal{CC}_1$ shown above. It checks whether the maximum limit of 5 users is reached by invoking the function "CardinalityR1" with value INCR indicating the addition of a user. If the maximum number of users for role R1 is not reached, then the role is activated, else the error "Maximum Number of Roles Reached" is raised by rule $\mathcal{CC}_1$. Similarly, when the role R1 is deactivated event E_4 is detected and the function "CardinalityR1" is invoked with DECR as the parameter, which reduces the count of the number of users active in role R1 by one so that new users can be activated.

Rule 5 (Check Access) *Check whether the subject (i.e., instance of $\mathcal{U}$) of a given session is or is not allowed to perform a given operation (e.g., read, write, etc.) on a given object (e.g., .dat file, etc.)*

```
EVENT    $E_6 = user \rightarrow checkAccess(sessionId, operation, object)$

RULE [   $\mathcal{CA}_1$
         ON     $E_6$
         WHEN  (sessionId IN sessionL) &&
                 (operation IN opsL) && (object IN objL) &&
                 (For ANY role IN getSessionRoles(sessionId)
                    (IF checkPermissions(operation,object, role) IS TRUE
                       return TRUE))
         THEN   < allow Access >
         ELSE    raise error "Permission Denied"   ]
```

Rule $\mathcal{CA}_1$ is triggered when the user tries to perform any operation on any object (i.e., event E_6). The rule allows the user requested operation only when at least one role from his *active role set* for that session has the required permission. This is carried out by the "For" statement that retrieves all the roles from the *active role set* and checks whether at least one role has the required permission using the "checkPermissions" function. Above shown rule is the same for all roles that do or do not take part in any type of relationships.

4.3.2 Rules Illustrating RBAC Extensions Enforcement

Even though RBAC has been extended extensively with various constraints, due to lack of space we demonstrate only temporal and control flow dependency constraints. Generalized Temporal RBAC [22, 23] provides an exhaustive set of temporal constraints. Control flow dependency constraints often occur in task oriented systems and are stricter forms of dependency constraints [23]. We show how a subset of time based SoD and post-condition constraints from [23] are supported.

Rule 6 (Disabling Time SoD) *Two roles from a given role set $\mathcal{RS}$ cannot be disabled at the same time in the interval (I, P) (NOTE: (I, P) corresponds to $\langle[begin, end], P\rangle$, where P is a periodic expression denoting an infinite set of periodic time instants, and [begin, end] is a time interval denoting lower and upper bounds that are imposed on instants in P). Role disabling is of main concern where availability is a primary concern [23]. For instance, both "Nurse" and "Doctor" roles cannot be disabled at the same time within the interval $([begin, end], P)$*[10].

```
EVENT    $StartD = event\ corresponding\ to\ date\ expression$
EVENT    $EndD = event\ corresponding\ to\ date\ expression$
EVENT    $ET_1 = roleDisableNurse()$
EVENT    $ET_2 = roleDisableDoctor()$
EVENT    $ET_3 = OR(ET_1, ET_2)$
EVENT    $ET_4 = Aperiodic([StartD], ET_5, [EndD])$
EVENT    $ET_5 = Aperiodic([10:00:00/*/*/*], ET_3,$
                          $[17:00:00/*/*/*])$

RULE [   $\mathcal{TSOD}_1$
         ON     $ET_4$
         WHEN  ( if roleDisableNurse == TRUE
                    ((if checkActiveDoctor() IS TRUE) return TRUE
                        else return FALSE)
                 else if roleDisableDoctor == TRUE
                    ((if checkActiveNurse() IS TRUE) return TRUE
                        else return FALSE))
         THEN   ( if roleDisableNurse == TRUE Then disableNurse()
                 else if roleDisableDoctor == TRUE Then disableDoctor())
         ELSE   ( if roleDisableNurse == TRUE
                        raise error "Denied as Doctor Already Disabled"
                 else if roleDisableDoctor == TRUE
                        raise error "Denied as Nurse Already Disabled"   ]
```

Rule $\mathcal{TSOD}_1$ provides time based SoD constraints, which does not allow both the roles "Nurse" and "Doctor" to be disabled at the same time in the interval (I, P). We have represented the interval I and P as [StartD, EndD] and ([10:00:00/*/*/*], [17:00:00/*/*/*]), but they can any type of simple or complex event. For example, [StartD] can be the start of the year and [EndD] can be the end of the year. Event ET_1 is raised whenever the role "Nurse" needs to be disabled. This will trigger the event ET_3 that is an OR event, which propagates the same to both the Aperiodic events ET_4 and ET_5. Event ET_4 triggers the rule $\mathcal{TSOD}_1$ when an user tries to disable the role Nurse within 10 a.m. and 5 p.m. and within [begin, end]. Rule $\mathcal{TSOD}_1$ determines whether the role "Nurse" can be disabled by checking whether the role "Doctor" is active, if so it allows to

[10]In this example we consider P as 10 a.m. to 5 p.m. every day. 10 a.m. every day is represented as [10:00:00/*/*/*], where the general form is "24h:mi:ss/mm/dd/yyyy". Periodic expression P in GTRBAC can be represented using event operators in active rules

disable role "Nurse" else it raises an error. In the similar way disabling of role "Doctor" is addressed. All the internal conditions such as checking with role lists are not shown in the above rule.

Rule 7 (Deactivating a Role after Δ) *Deactivate an activated role after a duration Δ. This is similar to limiting car parking to a fixed number of hours at one time [22]. Below shown rules restrict the duration constraints on a per user-role basis, where a role R3 is deactivated after the specified maximum duration in one activation by user Bob.*

```
EVENT    $ET_5 = Bob \rightarrow addActiveRoleR3(sessionId)$
EVENT    $ET_6 = startEventET_7(sessionId)$
EVENT    $ET_7 = PLUS(ET_6, \Delta)$

RULE [   $\mathcal{AAR}_5$
         ON     $ET_5$
         WHEN ...
         THEN  (...) startEvent$ET_7$(sessionId)
         ELSE   ...   ]

RULE [   $\mathcal{TSOD}_2$
         ON     $ET_7$
         WHEN  TRUE
         THEN  deactivateRoleR3(sessionId)   ]
```

Event ET_5 is raised whenever the user Bob activates the role R3. This triggers rule $\mathcal{AAR}_5$ which in turn raises the event ET_6. We have not shown all the other clauses of rule $\mathcal{AAR}_5$, intentionally. Event ET_6 starts the PLUS event ET_7. After the duration Δ event ET_7 is detected and rule $\mathcal{TSOD}_5$ is raised, which deactivates the role R3. Event ET_5 cannot be used to start the PLUS event ET_7 as event ET_7 should be started only after the role R3 is activated.

Rule 8 (Post-condition CFD) *If an event occurs, then the other event must also occur. For instance, if role "SysAdmin" role is enabled then role "SysAudit" must also be enabled, other wise both the roles should not be enabled.*

```
EVENT    $ET_8 = enableRoleSysAdmin()$
EVENT    $ET_9 = enableRoleSysAudit()$
EVENT    $ET_{10} = disableRoleSysAdmin()$
EVENT    $ET_{11} = disableRoleSysAudit()$

RULE [   $\mathcal{CFD}_1$
         ON     $E_8$
         WHEN  (...)
         THEN  (...) enableRoleSysAudit()
         ELSE   raise error "Cannot Activate SysAdmin"   ]

RULE [   $\mathcal{CFD}_2$
         ON     $E_9$
         WHEN  (...)
         THEN  (...)
         ELSE   disableRoleSysAdmin() &&
                  raise error "Cannot Activate SysAudit"   ]
```

As shown above, when the role "SysAdmin" has to be enabled, it will trigger event ET_8 which will trigger rule $\mathcal{CFD}_1$. This rule enables the role "SysAdmin" and tries to enable role "SysAudit" by triggering event ET_9. This event triggers rule $\mathcal{CFD}_2$ which in turn enables the role "SysAudit" when all conditions are met. When it cannot enable the role "SysAudit" it disables the role "SysAdmin" by invoking the function "disableRoleSysAdmin". As function "enableRoleSysAudit" is the only function that can enable the role "SysAudit" OWTE rules overcomes the problems faced by [23].

4.3.3 Rules Illustrating Active Security

Active security is critical as it detects and monitors the state changes of the underlying system to take timely actions.

Rule 9 (Transaction Based Activation) *Any junior employee is allowed to activate the role "JuniorEmp" ONLY IF the role "Manager" is activated. On the other hand, if the role "Manager" is deactivated, then role "JuniorEmp" should also be deactivated.*

```
EVENT    $ET_{12} = user \rightarrow addActiveRoleManager(sessionId)$
EVENT    $ET_{13} = user \rightarrow addActiveRoleJuniorEmp(sessionId)$
EVENT    $ET_{14} = user \rightarrow deactivateRoleManager(sessionId)$
EVENT    $ET_{15} = user \rightarrow deactivateRoleJuniorEmp(sessionId)$
EVENT    $ET_{16} = startEventET_{16}(sessionId)$
EVENT    $ET_{17} = startEventET_{17}(sessionId)$
EVENT    $ET_{18} = Aperiodic(ET_{16}, ET_{13}, ET_{17})$
 RULE [  $\mathcal{ASEC}_1$
         ON     $E_{12}$
         WHEN  (...)
         THEN  $< activateRoleManager >$
                 <startEvent$ET_{16}$(sessionId)>
         ELSE   raise error "Permission Denied"   ]

 RULE [  $\mathcal{ASEC}_2$
         ON     $E_{14}$
         WHEN  (...)
         THEN  $< deactivateRoleManager >$
                 if activeJuniorEmp() IS TRUE
                       $< deactivateRoleJuniorEmp >$
                 startEvent$ET_{17}$(sessionId)
         ELSE   raise error "Permission Denied"   ]

 RULE [  $\mathcal{ASEC}_3$
         ON     $E_{18}$
         WHEN  (...) return (TRUE|FALSE))
         THEN  $< activateJuniorEmp >$
         ELSE    raise error "Permission Denied"   ]
```

Event ET_{12} and ET_{13} are raised when roles "Manager" and "JuniorEmp" have to be activated, respectively. Similarly, event ET_{14} and ET_{15} are raised when the roles have to be deactivated. On the other hand, event ET_{16} is raised after role "Manager" is activated and event ET_{17} is raised after the role "Manager" is deactivated. As shown, event ET_{12} triggers rule $ASEC_1$ which in turn activates the role and raises event ET_{16}. Similarly, event ET_{14} triggers rule $ASEC_2$ which in turn deactivates the role and raises event ET_{17}.

Let us assume that a user is trying to activate role "JuniorEmp". It will raise event ET_{13}, which does not take any action as the Aperiodic event ET_{18} is not yet started. Activating role "Manager" triggers event ET_{12} which checks for the necessary conditions and activates the role, and raises event ET_{16} that in turn starts the Aperiodic event ET_{18}. Now, when an user tries to activate role "JuniorEmp", it raises event E_{13} which in turn raises the

Aperiodic event ET_{18} that has been already started. This triggers the rule $\mathcal{ASEC}_3$ which in turn checks whether all the constraints are satisfied in the "W" clause and returns TRUE or FALSE. If it returns TRUE, role "JuniorEmp" is activated using $< activateJuniorEmp >$. On the other hand, deactivating role "Manager" raises event ET_{14}. After deactivation, role "JuniorEmp" is deactivated and event ET_{17} is raised, which in turn terminates the Aperiodic event ET_{18}. As the event ET_{17} acts as a terminator it stops the Aperiodic event ET_{18} and the future activation of role "JuniorEmp", until the role "Manger" is activated again.

4.4 Summary and Advantages of OWTE Rules

In this section we have explained the synthesis of active authorization rules for enforcing RBAC and its extensions. We have demonstrated the seamless approach for supporting RBAC, its extensions and active security with various examples and rules. In addition to the above, privacy-aware RBAC [19] can also be enforced using OWTE rules as it also follows the Entity Relationship model described before.

5 Prototype Implementation

Sentinel [15, 12] is an active object oriented system that supports event based rule capability i.e., Event-Condition-Action model, using a uniform framework. In Sentinel, a reactive object is an object that has traditional object definition plus an event interface and a notifiable object is capable of being informed of the occurrence of some event. The event interface lets the object designate some or all of reactive object methods as primitive event generators. Together, both the kind of objects enable asynchronous communication with the rest of the system. Sentinel includes an event detector that is responsible for processing all the notifications from different objects and eventually signaling to the rules that some event has occurred triggering them. In addition, external monitoring module supports external events such as those from sensors, thus, supporting location/context-aware events. Sentinel+ is an enhanced version of Sentinel that supports OWTE rules. In our implementation, users are allowed to provide high level specification of enterprise access control policies (ACPs) using RBAC Manager[11], a graphical tool (i.e., a widget tool kit). In RBAC Manager, ACPs are specified using various widgets which takes the form of a Entity-Relationship like model as described in Section 2.

Enterprise XYZ described below will be used in this section to explain user specification, rule generation and implementation. In enterprise XYZ, ACPs are formulated using NIST RBAC with *static SoD with role hierarchies*. It consists of two major departments "purchase" and "approval". Purchase department is authorized to place the "purchase order" for equipments or other materials required by the enterprise. Approval department is the one that can authorize the purchases. Thus, static SoD relations are required, since the same person placing purchase orders cannot authorize it. In enterprise XYZ, there are five roles with the following hierarchies *purchase manager* (PM) $\rightarrow$ *purchase clerk* (PC) $\rightarrow$ *clerk*, and *approval manager* (AM) $\rightarrow$ *approval clerk* (AC) $\rightarrow$ *clerk*. For instance, role PM is a senior role to PC. Roles AC and PC have static SoD constraint relation between them. Since role hierarchies are present PM inherits the static SoD constraints from PC. Thus, a user assigned to the role PM cannot be assigned to the role AM or AC. Likewise, an user assigned to the role AM cannot be assigned to the role PM or PC.

[11]We don't explain the GUI due to space constraints. In essence, the policy shown in Figure 1 using a drag-n-drop mechanism.

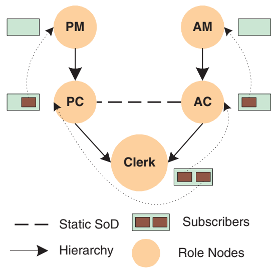

Figure 1. Access Control Policy Specification

High level specification for enterprise XYZ is shown in Figure 1. All the nodes represent an instance of entity $\mathcal{R}$ (i.e., roles). First, the role nodes corresponding to roles PM, AM, PC, AC and Clerk are created. Flags corresponding to relationships (i.e., hierarchy, station SoD relations, and active security constraints) are stored in the node. For instance, role nodes PC and AC have the Static SoD flags set once they are connected using the *dashed* line. All the flags are set when the role node to TRUE or FALSE when the policies are specified using a graphical tool RBAC Manager. Parent nodes are connected to the child nodes when there is a hierarchical relationship and static SoD constraints are represented as a dashed line between two nodes. Each node has an internal *subscriber* list that is used to point to the parent node. This pointer allow the child nodes to identify their parent nodes when the list of authorized users is required. On the other hand, constraints can be propagated in a bottom up manner using the pointers. (Note: Pointers shown in the Figure are not specified by users and they are generated by the system using the flags.)

Once the policies are specified, they are instantiated and the rules are generated. Let us take role PC for the discussion. Role PC has a static SoD and role hierarchy and when the policies are instantiated, rules corresponding to the role PC are generated. For instance, rule corresponding to activating role PC (i.e., "addActiveRolePC") is created. This rule is similar to rule $\mathcal{AAR}_2$ that was explained in Section 4 as role PC has static SoD and role hierarchies. Similarly all the other rules corresponding to PC and all the other roles are also created. Once all the rules are created it can enforce the policy specified by the enterprise. Currently, we assume that the policies specified using NIST RBAC and others do not have inconsistencies, but we are in the process of developing advanced consistency checking mechanisms.

When there is a change in the policy, for example, the shift time of role "day doctor" is changed from (8 a.m. to 4 p.m.) to (9 a.m. to 5 p.m.), it can be easily changed in the high level specification and the corresponding rules can be regenerated. This can be done without burdening the administrator as the rules are created using the access specification graph. With current systems and models it is a cumbersome process as all the low level semantic descriptions have to be changed manually. When there are thousands of rules, it is highly error prone to change them manually.

6 Related Work

Rule Processing Models: Event-Condition-Action rules [16, 10, 11, 1], to name a few, provide event-based active capability to the underlying system. Of all the event specification languages Snoop(IB) [11, 1] provides a rich set of operators and consumption modes not available in other languages. The rules can be either at the application level or the system level. Earlier work on application level ECA rules has been on making Relational Databases and Object Oriented Databases active capable. Similarly, ECA rules at the system level were used for providing various functionalities such as customizing the internal behavior of a database management system by realizing a number of transaction models and operating system load management. System level use of ECA rules can combine adaptive or self-monitoring capability with others such as the enforcement of RBAC.

Systems/Models Supporting RBAC: As explained in Section 1, none of the existing systems to our knowledge support the complete RBAC standard and its extensions in a seamless way. Thus, we explain some of the current systems below along with the features they support.

OASIS [32, 5] contains two types of rules and they are 1) activation rules, and 2) authorization rules. It supports dynamic role deactivations by use of rules, and it does not support role hierarchies explicitly and cardinality constraints. With the extended model, OASIS supports minimal temporal constraints, and context dependent constraints in the form of environmental predicates. OASIS requires administrators to specify enterprise policies using pseudo-natural language (i.e., Restricted English [4]) for authentication and authorization. These predicates are translated into a series of forms such as higher-order logic, first-order predicate calculus, horn clauses and finally converted to Java classes. The generated Java classes change as the authorization rules change. Even though OASIS uses pseudo-natural language that is transparent, it is cumbersome and a cognitive-burden for administrators. The implementation of OASIS is not clearly discussed except the fact that it takes a middle-ware approach. In this paper we have shown how OWTE rules can support role hierarchies and cardinality constraints.

Adage [30, 20], a rule-based authorization system for distributed applications, supports separation of duty by using history based constraints. The system does not support important RBAC features such as role hierarchies and cardinality constraints. It requires the administrators to specify the authorization rules manually. X-GTRBAC [8] is an XML based policy specification framework and architecture for enterprise wide access control. The framework supports GTRBAC specifications [22, 23] and enforces a set of GTRBAC constraints. The model does not support time based SOD. It requires administrators to specify policies using X-GTRBAC specification language based on a BNF-like grammar, called X-Grammar provided by the framework. In this paper we have shown how OWTE rules are used to support time based SoDs.

Temporal RBAC [6] "supports periodic role enabling and disabling, and temporal dependencies among such actions." Role triggers are used to express these temporal dependencies and to enable and disable roles either immediately or after a specified amount of time. On the other hand, in this paper we show how active rules are used to support RBAC, its extensions and active security, seamlessly. We have also shown how the Generalized Temporal RBAC [22, 23] can be enforced, using examples. [7, 24] show how multipolicy access control is supported but they do not explicitly consider the extended RBAC with various constraints and active security. [27] shows how context constraints are enforced in an RBAC environment.

Attribute-Based RBAC or AB-RBAC [2, 3] is a rule-based model that was developed to assign users to roles automatically, based on the authorization rules defined by enterprise administrators. Rule-based language defined in AR-RBAC supports minimal temporal aspects in the form of range constraints, but is not expressive enough to support various other constraints. In addition, this model also requires enterprise administrators to manually specify rules using the RB-RBAC language. Role hierarchies are induced from the seniority among authorization rule attributes when all the assumptions hold. In this paper we have shown how rules are created and maintained automatically from a high level specification.

7 Conclusions and Future Work

In this paper we have shown how active authorization rules or enhanced ECA rules are used to enforce RBAC, and its extensions such as temporal, and control flow dependency constraints in a seamless way. We have also shown how active security is provided that can take timely actions and that can prevent malicious activities. Large enterprises have hundreds of roles, which requires thousands of rules for providing access control, and generating these rules manually is error-prone and a cognitive-burden for non-computer specialists. High level specifications of access control policies eliminate the above problems and are transparent to non-computer specialists. OWTE rules are not created manually by administrators and we have shown briefly how authorization rules are generated from high level specifications of access control policies using Sentinel+. Rules generated have different granularities and classifications based on their functionalities. Automatic translation and maintenance of authorization rules is efficient and has less administrative burden. Expression of RBAC standard and its extensions in terms of OWTE rules provide practically applicable view of RBAC.

As the future work, various kinds of constraints that can be supported should be formalized, and the generated rules should be verified. It will be interesting to explore on how to generalize constraint specification in RBAC using events, to provide distributed access control for enterprises and to support other security models using OWTE rules.

References

[1] R. Adaikkalavan and S. Chakravarthy. SnoopIB: Interval-Based Event Specification and Detection for Active Databases. In *Proc. of ADBIS*. LNCS 2798, Sept. 2003.

[2] M. A. Al-Kahtani and R. Sandhu. A Model for Attribute-Based User-Role Assignment. In *Proc. of ACSAC*, 2002.

[3] M. A. Al-Kahtani and R. Sandhu. Induced Role Hierarchies with Attribute-Based RBAC. In *Proc. of SACMAT*, 2003.

[4] J. Bacon, M. Lloyd, and K. Moody. Translating Role-Based Access Control Policy Within Context. In *Proc. of POLICY*, 2001.

[5] J. Bacon, K. Moody, and W. Yao. A Model of OASIS Role-Based Access Control and its Support for Active Security. *TISSEC*, 5(4), 2002.

[6] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A Temporal Role-Based Access Control Model. *ACM TISSEC*, 4(3):191–233, 2001.

[7] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A System to Specify and Manage Multipolicy Access Control Models. In *Proc. of POLICY*, 2002.

[8] R. Bhatti. X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control. Master's thesis, Dept. of Electrical and Computer Science, Purdue University, 2003.

[9] R. Bhatti et al. X-GTRBAC Admin: A Decentralized Administration Model for Enterprise Wide Access Control. Technical Report 2004-04, CERIAS, Purdue University, 2004.

[10] A. P. Buchmann et al. *Rules in an Open System: The REACH Rule System*, pages 111–126. Springer, 1993.

[11] S. Chakravarthy et al. Composite Events for Active Databases: Semantics, Contexts, and Detection. In *Proc. of VLDB*, pages 606–617, 1994.

[12] S. Chakravarthy et al. Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules. *Information and Software Technology*, 36(9):559–568, 1994.

[13] T. M. Chalfant. Role Based Access Control and Secure Shell — A Closer Look At Two Solaris ™ Operating Environment Security Features, July 2003.

[14] R. Chandramouli and R. S. Sandhu. Role-Based Access Control Features in Commercial Database Management Systems. In *Proc. of NISSC*, 1998.

[15] R. Dasari. Events And Rules For JAVA: Design And Implemenation Of A Seamless Approach. Master's thesis, CIS Department, The University of Florida, Gainesville, 1999.

[16] U. Dayal, A. Buchmann, and S. Chakravarthy. *The HiPAC Project*. Morgan Kaufman Publishers Inc., 1996.

[17] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A Role Based Access Control Model and Reference Implementation within a Corporate Intranet. *TISSEC*, 1(2), 1999.

[18] D. F. Ferraiolo, J. A. Cugini, and D. R. Kuhn. Role-Based Access Control: Features and Motivations. In *Proc. of Computer Security Applications Conference*, 1995.

[19] Q. He. Privacy Enforcement with an Extended Role-Based Access Control Model. Technical Report TR-2003-09, Department of Computer Science, NCSU, 2003.

[20] J. Hoagland. Adage, 1999.

[21] InterNational Committee for Information Technology Standards. *RBAC Standard, ANSI INCITS 359-2004*, 2004.

[22] J. B. D. Joshi et al. Generalized Temporal Role-Based Access Control Model - Specification and Modeling. Technical Report 2001-47, CERIAS, Purdue University, 2001.

[23] J. B. D. Joshi et al. Dependencies and Separation of Duty Constraints in GTRBAC. In *Proc. of SACMAT*, 2003.

[24] M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the specification and evolution of access control policies. In *Proc.of SACMAT*, 2001.

[25] M. J. Moyer and M. Ahamad. Generalized Role-Based Access Control. In *Proc. of ICDCS*, 2001.

[26] National Institute of Standards and Technology (NIST). *The Economic Impact of Role-Based Access Control*, 2002.

[27] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. In *Proc. of SACMAT*, 2003.

[28] R. Ramakrishnan and J. Gehrke. *Database Management Systems (3rd ed.)*. McGraw-Hill, 2003.

[29] R. S. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[30] R. T. Simon and M. E. Zurko. Separation of Duty in Role-Based Environments. In *Proc. of IEEE CSF Workshop*, 1997.

[31] M. Strembeck. Conflict Checking of Separation of Duty Constraints in RBAC - Implementation Experiences. In *Proc. of the Conference on Software Engineering*, 2004.

[32] W. Yao, K. Moody, and J. Bacon. A Model of OASIS Role-Based Access Control and its Support for Active Security. In *Proc. of SACMAT*, 2001.