



Department of Computer Science and Engineering  
University of Texas at Arlington  
Arlington, TX 76019

# A FRAMEWORK FOR SUPPORTING AND ENFORCING RBAC AND ITS EXTENSIONS IN A SEAMLESS MANNER

Raman Adaikkalavan and Sharma Chakravarthy

Technical Report CSE-2004-2  
February 2004

# A FRAMEWORK FOR SUPPORTING AND ENFORCING RBAC AND ITS EXTENSIONS IN A SEAMLESS MANNER

Raman Adaikkalavan and Sharma Chakravarthy

*Computer Science and Engineering Department*

*The University of Texas at Arlington*

*Arlington, TX 76019, USA*

*{adaikkal, sharma}@cse.uta.edu*

**Abstract:** Role-Based Access Control (RBAC) has proven as a cost effective as well as a practical solution for authorization management in large enterprises. In the recent past, RBAC has been widely explored and there have been several extensions to it. Current systems do not enforce standard RBAC features and its extensions in a seamless way, which is essential to make RBAC even better-suited for a wide range of applications. In this paper, we propose an Event-Driven RBAC (ED-RBAC) framework that uses Event-Condition-Action (ECA) Rules for enforcing standard RBAC features and its extensions, such as the Generalized Temporal RBAC (GTRBAC) in a seamless way. Unlike other models, where authorization rules are defined by the enterprise, in our framework authorization rules are generated automatically from the enterprise security policy and are used for dynamic user-role assignment, seamless enforcement of diverse constraints, role deactivations, and so on. Automatic generation of authorization rules is indispensable, since thousands of rules are required for authorization management when there are hundreds of roles. In addition, conditions/constraints specification has been generalized so that this approach can support current and future extensions.

**Key words:** RBAC, Role-Based Access Control, ECA rules, Expressive constraint specifications, Authorization rules

## 1. INTRODUCTION

Role-Based Access Control (RBAC) [1, 2], where object accesses (or operations) are controlled by roles (or job functions) in an enterprise rather than a user or group, has proven as a positive alternative to traditional discretionary and mandatory access control. RBAC is role-centric, where users and objects are assigned to roles. Even though RBAC is not a panacea for access control issues, with its rich specification it has proven to be cost effective to enterprises by reducing the cost and complexity for authorization management of data. RBAC has been shown to be policy neutral, to support traditional access control mechanisms and MLS systems. RBAC [3] can be flat, hierarchical, and constrained and it has been extended with various constraints [4, 5] over the years making it more flexible as well as support diverse applications. These

extensions are in the form of temporal constraints, control flow dependent constraints, context-aware constraints, and so on. Specifications have played a predominant role compared to the enforcement of RBAC.

With the inherent nature of RBAC and authorization rules, current models/systems [6-9] use some form of authorization rules while enforcing RBAC. Enterprises using these models/systems for access control are required to manually write/define authorization rules. These rules can be used to dynamically assign users to roles based on the attribute values/credentials [7, 10], to apply Separation of Duty (SOD) constraints [8, 11], and so on. Large enterprises have hundreds of roles and in order to support these huge numbers of roles thousands of authorization rules must be defined by the enterprise/security administrators. Defining these authorization rules by the security administrators from the security policy is not feasible, since their complexity varies with enterprise's security policy. Even though roles are assumed to be static in RBAC, when they change, it will be difficult and error-prone for the enterprise administrators to modify the corresponding authorization rules. Thus, automatic creation and maintenance of these rules from security policies makes RBAC more suitable for enterprises.

Temporal constraints are crucial in access control to support time-based access control in various application domains such as health care, and workflow management systems (WFMSs). Generalized Temporal RBAC (GTRBAC) [4, 12] provides an exhaustive set of temporal and Control Flow Dependency (CFD) [5] constraints. Current models/systems support temporal constraints in a very limited form and are not generalized to enforce all the other constraints. When a system enforces RBAC it should be able to handle two key aspects and they are i) changes in the role structure, and ii) extensions to RBAC. Addressing first aspect requires rewriting of authorization rules that can be done either manually or automatically. Current systems require manual rewriting of these rules which is difficult and error-prone, whereas the framework presented in this paper supports automatic rewriting and maintenance of rules. Handling the second aspect requires a flexible and generalized framework so that it can support the future extensions in a uniform and seamless manner. The framework presented in this paper supports the enforcement of RBAC and its extensions in a seamless way.

In this paper, we propose an Event-Driven RBAC (ED-RBAC) framework based on the use of Event-Condition-Action (ECA) Rules for the enforcement of standard RBAC features and its extensions. Event-Condition-Action rules [13-15] provide event-based active capability to the underlying system. Whenever an event (E) occurs, the associated condition (C) is evaluated, and the corresponding action (A) is performed when the associated condition evaluates to true. In ECA rules, events are the occurrence of interest and they can be any arbitrary event (e.g., method invocations by an object, data from sensors, occurrence of patterns in text streams, and so forth). These rules can be defined either at the application level or at the system level. Earlier works on application level ECA rules have shown how to make passive (or non-active) object-oriented and relational databases active [14], and system level rules have been shown to support flexible transactions [16] in databases. In addition, we have developed many applications using ECA rules and those include, but not limited to, InfoFilter (A Text Stream Filtering System) [17], WebVigil (A Web Monitoring and Change Detection System) [18], and Alert Server (System for Publishing/Subscribing Alerts) [19]. Active capability has been extended to distributed applications as well [20, 21].

## **1.1 Our Contributions**

In this paper an ECA Rule-Based framework for enforcing RBAC is proposed. The key contributions of this paper are

- An ECA Rule-Based architecture for supporting standard RBAC and its extensions in a seamless manner.
- Authorization rules corresponding to the enterprise's security policy are generated and maintained automatically thereby avoiding manual intervention.
- This framework provides generalized specification of policies, expressive constraint/condition checking, seamless enforcement, uniformity in handling constraints, adaptability to the changes in the organization structure and RBAC extensions.

## **1.2 Outline**

The rest of the paper is as follows. Section 2 discusses the related work. Section 3 discusses RBAC, issues to be handled while enforcing RBAC and ECA rules. Section 4 discusses mapping of RBAC to ECA and generation of authorization rules. Section 5 describes the ED RBAC framework. Section 6 has conclusions and future work. Section 7 has references.

## **2. RELATED WORK**

OASIS [9, 22] supports dynamic role deactivations by use of rules and it does not support role hierarchies, cardinality constraints. With extended model, OASIS supports minimal temporal and context dependent constraints in the form of environmental predicates. OASIS requires administrators to specify enterprise policies using Restricted English [23], which is then translated into a series of forms such as higher-order logic, first-order predicate calculus and horn clause and then finally converted to Java classes. These generated Java classes change as the authorization rules change. The implementation of the OASIS is not clearly discussed except the fact that it takes a middleware approach.

Adage [8, 11], a rule-based authorization system for distributed applications, supports separation of duty by using history based constraints. This system does not support important RBAC features such as role hierarchies, and cardinality constraints. This system requires the administrators to specify the authorization rules manually.

Attribute-Based RBAC or AB-RBAC [7, 10] 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 is not expressive enough to support the constraints explicitly such as cardinality, control flow dependency constraints and prerequisite roles. It supports minimal temporal aspects in the form of range constraints. In addition, this model also requires the enterprise administrators to specify policies as authorization rules using the RB-RBAC language. Role hierarchies are induced from the seniority among authorization rule attributes when all the assumptions hold.

X-GTRBAC [6] is an XML based GTRBAC specification language which enforces a set of GTRBAC constraints. This model does not support the time based SOD. The system also requires the administrators to specify the policies using the XML format specified.

In essence, all the above-mentioned systems do not support some of the important features that large enterprises needs such as the temporal constraints, cardinality, and control flow dependency constraints. Current systems require enterprise administrators to specify the security policies in some form of authorization rules, which makes these systems inflexible. In addition, administrators are required to know these languages and the mapping of their enterprise security policy to these languages.

Event-Condition-Action rules [14, 15, 24-26] provide event-based active capability to the underlying system. Of all the event specification languages, Snoop [14, 15] provides a rich set of operators and contexts not available in other languages. Hence, we use the ECA paradigm proposed in Sentinel [14]. These rules can be either at the application level or at the system level. Earlier works on application level ECA rules have shown how to make Relational Databases and Object Oriented Databases active capable. In addition, there are many applications that have been developed using ECA rules and those include, but not limited to, WebVigiL (A Web Monitoring and Change Detection System) [18], InfoFilter (Text Filtering System) [17], and Alert Server (System for Publishing/Subscribing Alerts) [19]. Similarly, ECA rules at the system level were used to customize the internal behavior of a database management system by realizing a number of transaction models [16]. System level use of ECA rules can combine adaptive or self-monitoring capability with others such as the enforcement of RBAC.

### **3. EVENT DRIVEN RBAC ENFORCEMENT**

This section gives brief introduction to RBAC, issues involved in enforcing RBAC, overview of ECA rules, and why it should be used for enforcing RBAC.

#### **3.1 Role-Based Access Control (RBAC)**

In RBAC, object accesses are controlled by roles (or job functions in enterprises) rather than users or groups. RBAC is role-centric, where users and objects are assigned to roles, since roles are assumed to be static. With the proposed standard [3], RBAC can be flat (basic relations), hierarchical (role hierarchies), and constrained (i.e., SOD constraints and so on). The RBAC reference model is defined in terms of four model components (or functional packages), and they are Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD) Relations, and Dynamic Separation of Duty (DSD) Relations.

Core RBAC recognizes five administrative elements, users (U), roles (R), and permissions (P), where operations (OP) are performed on objects (OBJ). Users include human beings, application processes, software agents, robots, and etc. RBAC is role-centric, where role is the job function played by a user in an organization or a semantic construct, around which the access control policies are formulated. Permissions are the privileges given to roles to perform some operation on system objects. To access an object in a system that employs RBAC, a user should

be assigned to a role, which has the privileges to access that particular object. Core RBAC addresses the basic relations on flat roles, and they are user-role assignment relations, permission-role assignment relations, and role activations as part of user's session.

Most of the job functions (or roles) in enterprises overlap with each other, or in other words, users playing different roles have common set of permissions on certain objects. For example, in a hospital, some of the patient's records are accessible by all the specialists and doctors. These overlapping functions of enterprises are recognized by the way of role hierarchies (i.e., Hierarchical RBAC). Thus, when a role A inherits role B, all the permissions of role B are accessible via role A.

The principle of least privilege (or need-to-know privilege) plays an important role in access control, since it provides users no more privileges than is necessary to perform his/her job function, thus avoiding adverse effects in the system. These issues are addressed in RBAC using the SOD relations that can be either static SOD (SSD) or dynamic SOD (DSD). Constraints provided by these static and dynamic SOD differ in terms of *when* these constraints need to be applied in the system (i.e., at the time of accessing the objects or while assigning roles).

In addition, constrained RBAC is currently being extended extensively to support context-aware constraints, content-based constraints, temporal constraints, control-flow dependency constraints, etc. With these extensions, other components of RBAC such as the role hierarchies are also redefined. These extensions make RBAC more powerful and a practicable solution for enterprises in authorization management.

### **3.2 Issues in ED RBAC enforcement**

RBAC contains four standard functional packages along with additional features such as temporal constraints, [4, 27] control flow dependencies, [5] and so on. Set of features provided by core RBAC is required to be supported by systems that enforce RBAC and those that need to conform to the standard. Moreover, systems that support all the functional packages along with the extensions are befitting for enterprises managing authorization of data using RBAC.

Specification of access control policies using roles, role hierarchies and SOD constraint relations requires substantial effort by security administrators. These policies are expressed using the languages provided by the underlying system. In depth knowledge of the language provided by the underlying system is required to express these access control policies. Support for complex policies depends on the expressiveness of these languages and the framework used to enforce these policies. To a greater extent, enterprises are trying to exploit the use of RBAC to reduce the cost for authorization management. To satisfy the needs of these enterprises RBAC standard is being extended extensively supporting more diverse applications.

Just extending the RBAC specification is not adequate since it requires the underlying system to be extended or modified to support these changes. Furthermore the underlying system should have a generalized/flexible framework to support these extensions seamlessly. When the underlying system is not capable of adapting to the changes in a uniform way (i.e., requires extensions to the languages, etc.), it requires substantial effort from the security administrators to

use them. Thus, models used to enforce RBAC in systems should be powerful in the specification of access control policies, expressing and enforcing those policies. Moreover it should also support the current RBAC specification and the future extensions in a uniform way and be able to adapt to the changes in policies.

AB-RBAC [7, 10] shows the usage of authorization rules in dynamically assigning users to roles. Authorization rules can be used more effectively, such as the role activation/deactivation, role enabling/disabling, permission-role assignments, etc. In order to support constrained, flat and hierarchical RBAC in a seamless way, authorization rules that are used should be generic to handle all the aspects of RBAC such as the core features, role hierarchies, SOD constraints, cardinality constraints, prerequisite roles, temporal constraints, and control-flow dependencies. Moreover, automatic generation of rules from security policies is indispensable since defining rules manually is not efficient when there are hundreds of roles present in an enterprise.

### 3.3 ECA Rules

An ECA rule consists of three components E-C-A, where “E” an occurrence of interest (i.e., event), “C” is the condition that should be checked and “A” is an action that should be performed when the condition evaluates to true. Below shown is the rule specification (*rule\_spec*), where *rule\_name* is the name of the rule, *event\_name* is the name of the event, *condition\_function* is the function (or method) that should be invoked when the event occurs, and *action\_function* is the function (or method) that should be invoked when the *condition\_function* returns true. Events are specified using the specification (*event\_spec*) shown below, where *event\_name* is the name of the event that is based on a method invocation or on an event expression (formed using event operators) [15, 28]. As shown, *begin* of a method invocation (i.e., before a method is executed) or *end* of a method invocation (i.e., after a method is executed) or both can be treated as an event occurrences.

```
rule_spec ::= rule rule_name [event_name, condition_function, action_function]
```

```
event_spec ::= event event_modifier method_name  
| event event_name = event_exp
```

```
event_modifier ::= event_name  
| begin (event_name)  
| end (event_name)  
| begin (event_name) && end (event_name)
```

In general, rules can be specified as  
ON *event occurrence*  
CONDITION *check for condition*  
ACTION *perform the required action*

Compared to all the other components, event component of an ECA rule is considered important since it is event-centric. There has been a lot of work including [13, 15, 28], to make the event component more expressive, so that it can support any arbitrary event. In general,

events can be any occurrence of interest in an application or in a system. Events can be method execution by objects, data manipulations in Relational-DBMS, absolute or relative temporal events, external events (i.e., events from sensors, and so forth), occurrence of regular expressions or keywords in text streams and etc. These events that are predefined in the underlying system are known as primitive or simple events. On the other hand, there are composite or complex events, which are composed of more than one primitive or composite event using event operators [13, 15, 17, 28] such as AND, NOT, OR, Aperiodic, Periodic, Frequency, Plus, and Sequence. By using composite events, any kind of constraint on event occurrences can be expressed. For example, method execution by an object can be restricted within a time interval by combining the event (method execution by an object) with a temporal event. Similarly, cardinality (i.e., controlling the number of event occurrences) is also supported. An event occurrence triggers the rule that can be in the form of multiple rules, nested/cascaded rules, prioritized rules, and causality rules along with the coupling modes [29]. Rules can also be of class level or instance level. When a class level rule is set then all the event occurrences (i.e., method invocations by objects) of that class can trigger rule whereas an instance level rule is triggered when a particular object invokes the method/function.

## **4. ENFORCING RBAC USING ECA**

In the previous section, RBAC, ECA Rules and issues needed to handle while enforcing RBAC were explained. Below, we describe the use of ECA framework as a medium to enforce RBAC, mapping RBAC specification to ECA rules, and how the mapped authorization rules solve these issues.

### **4.1 RBAC to ECA Mapping**

With the nature of access control, all object (or system resource) accesses by the users through roles can be modeled as functions. RBAC standard provides the complete functional specification for “general administrative”/ “administrative review operations”, and “system supporting operations” for all the four components. GTRBAC provides the set of functions to support time-based access control in RBAC. As explained above, function or method invocations by the objects of a class are potential events in an ECA rule-based system. User invocations of functions/methods for accessing objects (or system resources) in an RBAC can be treated as event occurrences. So user-role assignment, permission-role assignment, changes in the role’s privileges on objects, changes in the user-role assignment, role activations/deactivations, role enabling/disabling, and so forth are potential events in RBAC. Thus, all the occurrences of interest in an RBAC system can be treated as events. Whenever an event is detected, the corresponding information relating to that event such as the context in which it is detected, location of detection, time of detection, and etc., are treated as properties/parameters of those events.

Standard RBAC and its extensions provide various constraints that control object accesses. Systems enforcing RBAC should be able to handle these constraints (e.g., temporal, cardinality, prerequisite roles, control flow dependency, SOD, context-aware, and so on). Current systems/models are customized to some particular domain and do not enforce all these constraints

in a generalized way. Moreover these systems should be modified to adhere to the changes/extensions in specifications. When ECA authorization rules are used, the C (condition) component can handle any arbitrary condition/constraint and it can be checked on event occurrences. Event properties/parameters that are made available by event occurrences are used in the process of constraint checking. On event occurrences corresponding constraints are checked and object accesses are allowed (action component) when conditions are met.

As mentioned above, ECA rules are generic and can handle arbitrary events, conditions and actions. With this expressive event specification, ECA rule based (ED RBAC) framework suites well for enforcing constrained, flat and hierarchical RBAC. Another advantage of using ECA rules is that the underlying system is made active, so that if there is any change in the system state with respect to the user properties it is pushed on to the system enabling automatic activation/deactivation. In essence the mapping of RBAC to ECA is shown in Table 1.

Table 1. RBAC to ECA mapping

RBAC	ECA Authorization Rules
User invocation of methods, (Role assignment, Object access, and etc.), Absolute or relative time point, etc.	Events
Constraints	Conditions
Allowing object access	Actions

## 4.2 Automatic generation of rules

Large enterprises with hundreds of roles require thousands of rules to enforce authorization management of data. Even though roles are assumed to be static, when there is a change in role structure there can be many rules that need to be changed. Rewriting or deleting these rules requires significant effort from the security administrators. Once rules are written they can be used in automatic assigning of users to roles with generalized constraints. Current systems/models have shown that rules are appropriate for enforcing SOD constraints, role activation, role deactivation, assignment of users to roles, and so on. Apart from these, in this approach rules can also be used to support more constraints such as cardinality, context-aware, prerequisite roles, and so on.

Consider the following examples:

**Simple RBAC policy:** *User Jack can be assigned to cashier role*

ON *Jack* → *AssignUser (cashier)*

CONDITION “*if not assigned to cashier role*”

ACTION “*assigns to cashier role*”

In the above rule, whenever user Jack invokes the function AssignUser he is assigned to the cashier role.

**RBAC policy with separation of duty constraint on user-role assignment:** *User Jim can be assigned to roles (cashier, accountant). Roles cashier and accountant are mutually exclusive.*

ON *Jim*  $\rightarrow$  *AssignUser (accountant)*  
CONDITION (if not active in cashier role and not active in accountant role)  
ACTION “assigns to accountant role”

ON *Jim*  $\rightarrow$  *AssignUser (cashier)*  
CONDITION (if not active in cashier role and not active in accountant role)  
ACTION “assigns to cashier role”

In the above two rules, whenever user Jim invokes the function AssignUser, he is assigned to the accountant role when he is not active in cashier role or vice versa

**RBAC policy with temporal and location constraint on user-role assignment:** *User Bob can be assigned to cashier role. This role is enabled only between 10 a.m. and 11 a.m. and in Arlington, TX (location)*

ON *Bob*  $\rightarrow$  *AssignUser (cashier)*  
CONDITION if (time “between 10 a.m. and 11 a.m.”) and (location “Arlington, TX”) and “not active in cashier role”  
ACTION “assigns to cashier role”

In the above rule, whenever user Bob invokes the function AssignUser he is assigned to the cashier role based on the temporal and location constraint.

Invocation of the functions (e.g., AssignUser ()) is an event occurrence. Additional parameters such as role, time, location, and so on are passed along with the function invocation to check the required constraints. For example, assigning user Bob to role cashier is done using the function call *Bob*  $\rightarrow$  *AssignUser (cashier, 10.12 a.m., “Arlington, TX”)*.

**Policy Specification using composite event operator:** Above policy can also be specified using the composite event operator *Aperiodic (E<sub>1</sub>, E<sub>2</sub>, E<sub>3</sub>)*, where E<sub>2</sub> is an event that should occur between the occurrence of E<sub>1</sub> (event that starts the composite event) and E<sub>3</sub> (event that starts the composite event). Thus the RBAC above policy can be specified as *Aperiodic (10 a.m., AssignUser (), 11 a.m.)*. In this, events E<sub>1</sub> and E<sub>3</sub> are primitive or simple absolute temporal event (i.e., 10 a.m.). Event E<sub>2</sub> is detected whenever the method AssignUser () is invoked by users. With this composite event, a user can be assigned to the role only between 10 a.m. and 11 a.m. When it is 11 a.m. the role deactivation/disabling can be done automatically.

Let us assume that user Bob is already assigned to the role cashier at 10.12 a.m. In the above policy, user Bob should not be allowed to access objects when the constraints are not met (i.e., after 11 a.m.). These can be handled in two ways and they are 1) deactivate the user from the corresponding role when the constraint is not valid (i.e., deactivation of role at 11 a.m. automatically), 2) when the user tries to access object check for the constraint and disallow access and deactivate. Since both ways are allowed in this framework the rules will generated according to enterprise policy. Below is the rule that is used to deactivate the user automatically when the temporal constraint is violated

ON *11 a.m.*  
CONDITION (if user Bob is active in role cashier)  
ACTION “deactivate the user Bob from role cashier (DeassignUser)”

ECA rules are powerful and are used as authorization rules in the enforcement of RBAC. As shown above, for the same RBAC policy there can be different rules and it is decided by the enterprise policy. Policy specifications are provided via a graphical user interface or an XML policy file. Initial set of rules are generated automatically for the provided specification. Modification in the policy requires modifications to the authorization rules and it is handled without manual intervention in this framework. If there is a conflict while rewriting the rules because of an active user, it is notified to the administrator and the corresponding conflicts are resolved.

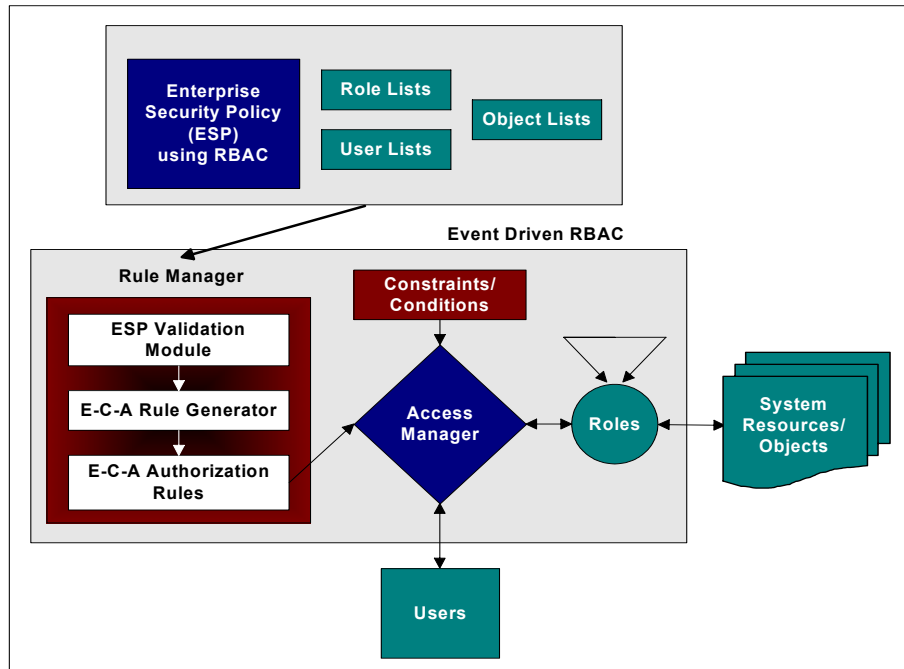


Figure 1. ED RBAC Framework

## 5. ED RBAC FRAMEWORK

Figure 1 shows the Event Driven RBAC framework. This framework handles policy specification, policy validation (with respect to role lists, user lists, and object lists), rule generation, and rule maintenance. It supports standard RBAC and its extensions with expressive constraint specification (temporal, cardinality, prerequisite roles, context-aware, and control-flow dependent constraints), automatic role enabling/disabling, role activation/deactivation, and user-role assignment. All the functionalities supported in this framework are based on the event occurrences (i.e., Event Driven) generated by the users (i.e., invocation of methods), time instances (i.e., temporal events) in the underlying system, etc.

Enterprise Security Policy (ESP) along with the role list, user list and object list is provided by the enterprise. These lists contain all the users, roles and objects (i.e., system resources) that are available in the enterprise. This policy specification can be of RBAC standard specification or its extensions and is provided by the security administrators using the graphical user interface or a XML file. Once policy specifications are made available, these policies are validated in the ESP

validation module in the rule manager for the proper existence of users, roles and objects using the lists available. ECA rule generator module generates the initial set of authorization rules used to enforce RBAC from the validated policies.

Once authorization rules are created, these rules are used by the access manager in ED RBAC to control the object accesses. Users, based on their object accesses, are automatically assigned to their respective roles by the access manager based on the conditions/constraints. Since ECA authorization rules have expressive constraint specification, they can handle diverse constraints on user accesses and assignments. Once assigned, access rights for objects/system resources are given by the access manager to the users based on their roles and constraints. When there is a change in the enterprise specification policy, authorization rules are rewritten and maintained by the rule manager. This is done without manual intervention when there is no conflict.

## **6. CONCLUSIONS AND FUTURE WORK**

This paper provides a generalized framework to enforce standard RBAC features and its extensions in a seamless way using the event driven approach. Enterprise security policy provided can be of RBAC or its extensions and this approach can handle constraints (e.g., temporal, cardinality, prerequisite roles, control flow dependency, separation of duty, context-aware, and so on) in a generalized way. Large enterprises with hundreds of roles require thousands of rules to enforce authorization management of data and this framework supports the automatic creation and maintenance of authorization rules. This framework also supports of the automatic user-role assignment, role activation, role deactivation, and so on. Currently this framework is being implemented using Local Event Detector [30] as the underlying mechanism.

## **7. REFERENCES**

1. Ferraiolo, D.F., J.A. Cugini, and D.R. Kuhn. *Role Based Access Control: Features and Motivations*. in *Computer Security Applications Conference*. 1995.
2. Sandhu, R.S., et al., *Role-Based Access Control Models*. IEEE Computer, 1996. **29**(2).
3. *NIST RBAC Standard (Proposed)*. 2003, National Institute of Standards and Technology (<http://csrc.nist.gov/rbac/rbac-std-ncits.pdf>).
4. Joshi, J.B.D., et al., *Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I) - Specification and Modeling*. 2001, CERIAS Technical Report 2001-47, Purdue University.
5. Joshi, J.B.D., et al. *Dependencies and Separation of Duty Constraints in GTRBAC*. in *ACM SACMAT*. 2003. Como, Italy.
6. Bhatti, R., et al., *X-GTRBAC Admin: A Decentralized Administration Model for Enterprise Wide Access Control*. 2004, CERIAS Technical Report 2004-04, Purdue University.
7. Al-Kahtani, M.A. and R. Sandhu. *A Model for Attribute-Based User-Role Assignment*. in *ACSAC*. 2002.
8. Simon, R.T. and M.E. Zurko. *Separation of Duty in Role-Based Environments*. in *IEEE Computer Security Foundations Workshop*. 1997.
9. Yao, W., K. Moody, and J. Bacon. *A model of OASIS role-based access control and its support for active security*. in *ACM SACMAT*. 2001. Chantilly, Virginia, USA.
10. Al-Kahtani, M.A. and R. Sandhu. *Induced Role Hierarchies with Attribute-Based RBAC*. in *ACM SACMAT*. 2003. Como, Italy.
11. Hoagland, J., *Adage*. 1999, <http://seclab.cs.ucdavis.edu/secsem2/01-20-99.pdf>.

12. Joshi, J.B.D., et al., *Generalized Temporal Role Based Access Control Model (GTRBAC) (Part II) - Expressiveness and Design Issues*. 2003, CERIAS Technical Report 2003-01, Purdue University.
13. Chakravarthy, S., et al., *Composite Events for Active Databases: Semantics, Contexts and Detection*, in *Proc. Int'l. Conf. on Very Large Data Bases VLDB*. 1994: Santiago, Chile. p. 606--617.
14. Chakravarthy, S., et al., *Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules*. Information and Software Technology, 1994. **36**(9): p. 559--568.
15. Adaikkalavan, R. *SnoopIB: Interval-Based Event Specification and Detection for Active Databases*. in *Advances in Databases and Information Systems*. 2003. Germany: Lecture Notes in Computer Science 2798.
16. Anwar, E., S. Chakravarthy, and M. Viveros, *An Extensible Approach to Realizing Advanced Transaction Models*, in *Advanced Transaction Models and Architectures*, S. Jajodia and L. Kerschberg, Editors. 1997, Morgan Kaufmann.
17. Elkhalfifa, L., R. Adaikkalavan, and S. Chakravarthy, *InfoFilter: Complex Pattern Specification and Detection Over Text Streams*. 2004, Technical Report CSE-2004-1 (<http://www.cse.uta.edu/Research/Publications/Downloads/CSE-2004-1.pdf>), Department of Computer Science and Engineering, The University of Texas at Arlington.
18. Pandrangi, N., et al. *WebVigiL: User Profile-Based Change Detection for HTML/XML Documents*. in *Twentieth British National Conference on Databases*. Coventry, UK, pages 38 - 55, 2003.
19. Vontela, N., *A PERSISTENT AND RECOVERABLE MIDDLEWARE APPROACH TO ALERT DISTRIBUTION*, in *M.S. Thesis* (<http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Nishanth.pdf>), Computer Science and Engineering, The University of Texas at Arlington. 2002.
20. Chakravarthy, S. and H. Liao. *Asynchronous Monitoring of Events for Distributed Cooperative Environments*. in *CODAS 2001*. 2001. Beijing: IEEE Computer Society.
21. Tanpisut, W., *Design and Implementation of Event based subscription/notification paradigm for distributed environments*, in *M.S. Thesis* ([http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Weera\\_thesis.pdf](http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Weera_thesis.pdf)), Computer Science and Engineering, The University of Texas at Arlington. 2001.
22. Bacon, J., K. Moody, and W. Yao, *A model of OASIS role-based access control and its support for active security*. ACM Transactions on Information and System Security (TISSEC), 2002. **5**(4): p. 492--540.
23. Bacon, J., M. Lloyd, and K. Moody. *Translating role-based access control policy within context*. in *In Policy 2001, Workshop on Policies for Distributed Systems and Networks*. 2001. Bristol, UK: Springer LNCS.
24. Buchmann, A.P., et al., *Rules in an Open System: The REACH Rule System*, in *Rules in Database Systems.*, N. Paton and M. Williams, Editors. 1993, Springer. p. 111--126.
25. Gatzju, S. and K.R. Dittrich, *SAMOS: an Active, Object-Oriented Database System*. in *IEEE Quarterly Bulletin on Data Engineering*, 1992. **15**(1-4): p. 23--26.
26. Gehani, N. and H.V. Jagadish, *Ode as an Active Database: Constraints and Triggers*, in *Proc. 17th Int'l Conf. on Very Large Data Bases*. 1991: Barcelona. p. 327-336.
27. Joshi, J.B.D., et al. *Temporal Hierarchies and Inheritance Semantics for GTRBAC*. in *ACM SACMAT*. 2002. Monterey, California, USA.
28. Chakravarthy, S. and D. Mishra, *Snoop: An Expressive Event Specification Language for Active Databases*. Data and Knowledge Engineering, 1994. **14**(10): p. 1--26.
29. Dayal, U., A. Buchmann, and S. Chakravarthy, *The HiPAC Project*, in *Active Database Systems - Triggers and Rules For Advanced Database Processing*. 1996, Morgan Kaufman Publishers Inc. p. 177--206.
30. Dasari, R., *Events And Rules For JAVA: Design And Implemenation Of A Seamless Approach*, in *Master's Thesis, Database Systems R&D Center, CIS Department*. 1999, University of Florida: Gainesville.