



# SESAME: Scalable, Environment Sensitive Access Management Engine

GUANGSEN ZHANG and MANISH PARASHAR

*The Applied Software Systems Laboratory (TASSL), Dept. of Electrical and Computer Engineering, Rutgers University, 94 Brett Road, Piscataway, NJ 08854*

**Abstract.** As computing technology becomes more pervasive and mobile services are deployed, applications will need flexible access control mechanisms. Although lots of researches have been done on access control, these efforts focus on relatively static scenarios where access depends on identity of the subject. They do not address access control issues for pervasive applications where the access privileges of a subject not only depend on its identity but also on its current context and state. In this paper, we present the SESAME dynamic context-aware access control mechanism for pervasive applications. SESAME complements current authorization mechanisms to dynamically grant and adapt permissions to users based on their current context. The underlying dynamic role based access control (DRBAC) model extends the classic role based access control (RBAC). We also present a prototype implementation of SESAME and DRBAC with the Discover computational collaboratory and an experimental evaluation of its overheads.

**Keywords:** security, access control, context-aware, pervasive computing, role based

## 1. Introduction

Pervasive computing and communication technologies are rapidly weaving themselves into the fabrics of everyday life and have the potential for fundamentally redefining the way we interact with information, each other, and the world around us. The proliferation of smart gadgets, mobile devices, PDAs and sensors has enabled the construction of pervasive computing environments, transforming regular physical spaces into intelligent spaces [7]. Such intelligent spaces provide services and resources that users can access and interact with via personal portable devices such as a PDA using short-range wireless communications such as Bluetooth or IEEE 802.11. The resulting anytime-anywhere access infrastructures is enabling a new generation of applications that can leverage this pervasive information Grid to continuously manage, adapt and optimize. One example of such an application is the Aware Home project at Georgia Institute of Technology [10]. Sensors in the home can capture, process and store a variety of information about its residents and their activities, enabling the Aware Home application to detect and respond to events in the room. Another application is the Intelligent Room project at MIT. In this application, computers are embedded in a room so that people can interact with computers the way they do with other people, using speech, gesture, movement and context [14]. Other applications are described in [4, 9]. Such pervasive applications are characterized by continuous pervasive access to information, resources and services and ad hoc, dynamic interactions between participating entities, and lead to significant research challenges.

One key challenge in pervasive applications is managing security and access control. Access Control List (ACL) is a very commonly used access control mechanism. In this approach, permission to access resources or services is moderated by checking for membership in the access control list associated with each object. However, this strategy is inadequate for pervasive applications as it does not consider context information.

In a pervasive environment, users are mobile and typically access resources (information, services, sensors, etc.) using mobile devices. As a result the context of a user (i.e. location, time, system resources, network state, network security configuration, etc.) is highly dynamic, and granting a user access without taking the user's current context into account can compromise security as the user's access privileges not only depend on "who the user is" but also on "where the user is" and "what is the user's state and the state of the user's environment". As a result, even an authorized user can damage the system as the system may have different security requirement within different contexts. Traditional access control mechanisms such as access control list break down in such an environments and a fine-grained access control mechanism that changes the privilege of a user dynamically based on context information is required.

Although a lot of work has been done in the area of access control, most of this work is user-centric, where only credentials of the user are considered when granting access permission. Relatively little research has been done to combine context information with credentials while making access control decisions. The existing research however does not address pervasive applications where context is dynamic and a user's privileges must continuously adapt based on the context.

This paper presents a dynamic context-aware access control mechanism that dynamically grants and adapts permissions to users according to current context. The proposed mechanism extends the role based access control (RBAC) model [3], while retaining its advantages (i.e. ability to define and manage complex security policies). The model dynamically adjusts *Role Assignments* and *Permission Assignments* based on context information. In our approach, each user is assigned a role subset (by the authority service) from the entire role set. Similarly the resource has permission subsets for

each role that will access the resource. During a secure interaction, state machines are maintained by delegated access control agents at the subject (*Role State Machine*) to navigate the role subset, and the object (*Permission State Machine*) to navigate the permission subset for each active role. The state machine consists of state variables (role, permission), which encode its state, and commands, which transform its state. These state machines define the currently active role and its assigned permissions and navigate the role/permission subsets to react to changes in the context.

The rest of this paper is organized as follows: Section 2 presents background and related work. Section 3 outlines a motivating application. Section 4 presents the proposed dynamic context-aware access control model. Section 6 presents a short discussion about the model and its implementation. Section 7 concludes the paper.

## 2. Background and related work

Role based access control (RBAC) [3, 16] is an alternative to traditional discretionary (DAC) and mandatory access control (MAC). In RBAC, users are assigned roles and roles are assigned permissions. A principle motivation behind RBAC is the ability to specify and enforce enterprise specific security policies in a way that maps naturally to an organization's structure. As user/role associations change more frequently than role/permission associations, in most organizations, RBAC results in reduced administrative costs as compared to associating users directly with permissions. It can be shown that the cost of administrating RBAC is proportional to  $U + P$  while the cost of associating users directly with permissions is proportional to  $U * P$ , where  $U$  is the number of individuals in a role and  $P$  is the number of permissions required by the role. Sandhu et al. [3, 16] define a comprehensive framework for RBAC models which are characterized as follows:

- $RBAC_0$ : the basic model with users associated with roles and roles associated with permissions.
- $RBAC_1$ :  $RBAC_0$  with role hierarchies.
- $RBAC_2$ :  $RBAC_1$  with constraints on user/role, role/role, and/or role/permission associations.

Recently RBAC was found to be the most attractive solution for providing security features in different distributed computing infrastructure [16]. Although the RBAC models vary from very simple to pretty complex, they all share the same basic structure of subject, role and privilege. Other factors such as relationship, time and location, which may be part of an access decision, are not considered in making access control decision in these models. In this paper, we extend  $RBAC_0$  to provide context-aware access control mechanisms for pervasive applications.

Giuri and Iglío [5] have proposed a role-based access control model that provides special mechanisms for the definition of content-based access control policies. By extending the notion of permission, they have allowed for the specification of

security policies in which the permission of an object may depend on the content of the object itself. For example, in a health-care organization, the physician is only allowed to access and modify patient records related to his or her patient.

Woo and Lam [19] designed a distributed authorization service using their Generalized Access Control Language (GACL). In their design, they use the notion of system load as the determining factor in certain access control decisions, so that, for example, certain programs can only be executed when there is enough system capacity available.

Finally, Michael J. Covington et al. [10, 11] have proposed the Generalized Role Based Access Control (GRBAC) model. In this model, they extend the traditional RBAC by applying the roles to all the entities in a system. (In RBAC, the role concept is only used for subjects). By defining three types of roles, i.e., Subject roles, Environment roles, and Object roles, GRBAC uses context-information as a factor in making access decisions.

All of the research efforts described above take additional factors into consideration when make access control decision. However, both Giuri, Iglío and Woo, Lam don't consider context information as a key factor in their access control mechanism. In GRBAC, the definition of environment roles allows the model to partially address problem we described, but it may not be feasible in practice because the potential large amount of environment roles make the system hard to maintain. Also, by defining too many roles in the system, it loses the advantage that RBAC provides.

## 3. Access control challenges for pervasive applications

To illustrate the motivation of our research, let us discuss an example application that will be enabled by a pervasive computing infrastructure in a smart building of a university, as illustrated in figure 1. The building has many rooms including faculty offices, administration offices, conference rooms, classrooms and laboratories. Sensors in the building can capture, process and store a variety of information about the building, the users and their activities. Pervasive applications in such an environment allow faculty, staff, students and administrators

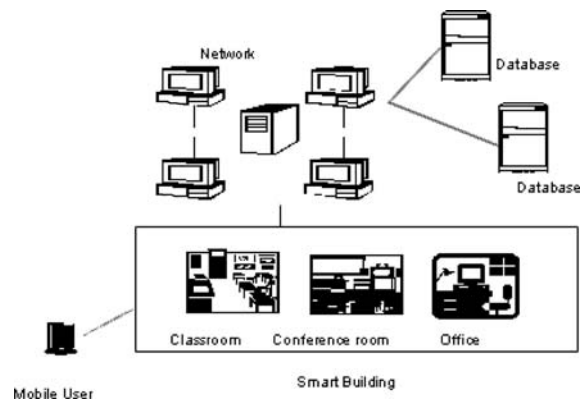


Figure 1. Smart building application.

to access resources/information from any locations at anytime while inside this building using mobile devices (PDAs) and wireless networks. While user credentials are still the basis for all the access control decisions, user's context information and application state should also be considered. For example, a student can only control the audio/video equipment in a classroom if she/he is scheduled to present in that class at that time by the faculty in charge. Similarly the payroll server should not be allowed to access if its load is above 80% or if the access is over an insecure link. In such applications, privileges assigned to the user will change as context changes. If the user is accessing the resource while the user's context information is changing (say the moves from a secure network link to an insecure link), specific access control mechanisms are needed to ensure that system/application security and consistency are maintained without decreasing flexibility.

The examples above embody many of the key ideas of the research presented in this paper. To maintain system security for such a pervasive application, we have to dynamically adapt access permissions granted to users as context information for the session changes. Context information here includes environment of the user such as location, time that the user access the resource and system information such as CPU usage and network bandwidth. The traditional RBAC models [3] do not directly address the requirements of such an application. In the RBAC model, the user is assigned a subset of roles when the user begins a session. This subset of roles are then used to access resources. During a session, although roles can be activated or deactivated based on constraints such as role conflict or prerequisite roles, the user's access privilege is not changed based on context information. Recently, Michael J. Covington et al., have proposed the GRBAC model [11] that used context to provide access control for Aware Home applications. However, the definition of environment role is not feasible for pervasive applications as described in the previous section.

#### 4. Dynamic role based access control model

Dynamic Role Based Access Control model (DRBAC) addresses the dynamic access control requirement of applications in pervasive environments. It extends the traditional Role Base Access Control (RBAC) model to use dynamic context information while making access control decision. Specifically, DRBAC addresses two key requirements motivated by the application in Section 3: (1) A user's access privileges must change when the user's context changes. (2) A resource must adjust its access permission when its system information (e.g., network bandwidth, CPU usage, memory usage) changes. In this section, we first formally define DRBAC and then describe its operation.

##### 4.1. DRBAC Definition

The DRBAC definition is based on the RBAC formalism presented in [8]. DRBAC has the following components:

- **USERS.** A user is an entity whose access is being controlled. **USERS** represents a set of users.
- **ROLES.** A role is a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role. **ROLES** represents a set of roles.
- **PERMS.** A permission is an approval to access one or more RBAC protected resources. **PERMS** represents a set of permissions.
- **ENVS.** **ENVS** represent the set of context information in the system. We use an authorized "Context Agent" to collect context information in our system.
- **SESSIONS.** A session is a set of interactions between subjects and objects. A user is assigned a set of roles during each session. The active role will be changed dynamically among the assigned roles for each interaction. **SESSIONS** represents a set of sessions.
- **UA.** **UA** is the mapping that assigns a role to a user. In the session, each user is assigned a set of roles, the context information is used to decide which role is active. The user will access the resource with the active role.
- **PA.** **PA** is the mapping that assign permissions to a role. Every role that has privilege to access the resource is assigned a set of permissions, and the context information is used to decide which permission is active for that role.

The model is illustrated in figure 2. In the approach, a Central Authority (CA) maintains the overall role hierarchy. When the user logs on the system, based on the user's capability, a subset of the role hierarchy is assigned to the user for each session. Then the CA sets up an agent for that user and delegates the user's right to that agent. The agent will monitor the environment status of the user and dynamically change the active role of the user. Every resource maintains a set of permission hierarchies for each potential role that will access the resource. The resource maintains its environment and dynamically adjusts the permissions for each role. We summarize the above discussions below:

DRBAC Definition:

- **USERS, ROLES, PERMS, ENVS and SESSIONS** (*users, roles, permissions, environments and sessions, respectively*).

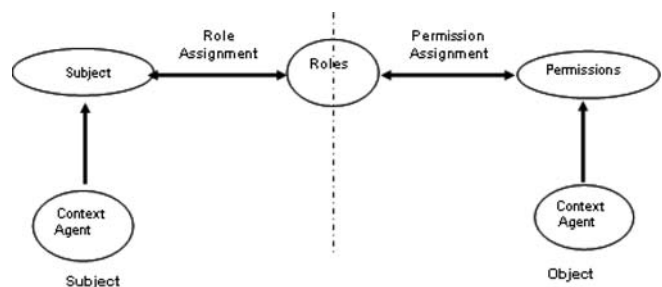


Figure 2. Dynamic access control model.

- *ACT\_ROLE* and *ACT\_PERMISSION* (active role and active permission respectively).
- $UA \subseteq USERS \times ROLES$ , a many-to-many mapping user-to-role assignment relation.
- $PA \subseteq PERMS \times ROLES$ , a many-to-many mapping permission-to-role assignment relation.
- *Assigned\_roles* ( $u:USERS, e:ENVS$ )  $\rightarrow 2^{ROLES}$ , the mapping of user  $u$  onto a set of roles.
- *Assigned\_permissions* ( $r:ROLES, e:ENVS$ )  $\rightarrow 2^{PERMS}$ , the mapping of role  $r$  onto a set of permissions.
- *User\_sessions* ( $u:USERS$ )  $\rightarrow 2^{SESSIONS}$ , the mapping of user  $u$  onto a set of sessions.
- *Session\_roles* ( $s:SESSIONS$ )  $\rightarrow 2^{ROLESS}$ , the mapping of session  $s$  onto a set of roles. Formally:  $session\_roles(s_i) \subseteq \{r \in ROLES \mid (session\_roles(s_i), r) \in UA\}$
- $RH \subseteq ROLES \times ROLES$  is a partial order on *ROLES* called the inheritance relation, written as  $\geq$ , where  $r_1 \geq r_2$  only if all *PERMS* of  $r_2$  are also *PERMS* of  $r_1$ , and all users of  $r_1$  are also users of  $r_2$ .
- $PH \subseteq PERMS \times PERMS$  is a partial order on *PERMS* called the inheritance relation, written as  $\geq$ , where  $p_1 \geq p_2$  only if all permissions of  $p_2$  are also permissions of  $p_1$ , and all roles of  $p_1$  are also roles of  $p_2$ .

#### 4.2. DRBAC explained

In DRBAC, each user is assigned a role subset from the entire role set. Similarly, each resource will assign a permission subset from the entire permission set to each role which has a privilege to access the resource. Figure 3 illustrates the relationship between the role hierarchy maintained at the Central Authority (CA) and the role hierarchy assigned to a particular user. It can be seen that the role hierarchy a user is a subset of the overall role hierarchy.

State machines maintain the role subset for each user and the permission subset for each role. A state machine consists of state variables, which encode its state, and events, which transform its state. In DRBAC, there is a Role State Machine for each user, and a Permission State Machine for each role. The role and permission are used as state variables respectively. The Context Agent collects context information and generates pre-defined events to trigger transitions in the state

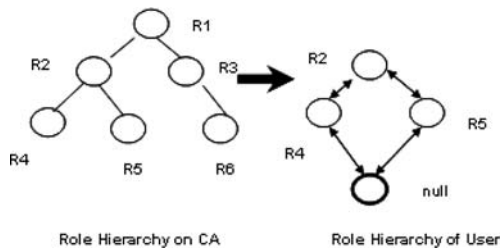


Figure 3. Role hierarchy state machine.

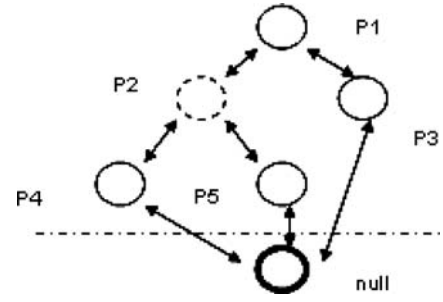


Figure 4. Permission hierarchy state machine.

machines. A permission state machine is illustrated in figure 4.

A null permission implies no permission. A transition is defined as  $T(\text{Initial State}, \text{Destination State})$ . So  $T(P1, P2)$  represents the transition from  $P1$  to  $P2$  and  $T(P2, P1)$  represents the transition from  $P2$  to  $P1$ . The Role State Machine is similar to the Permission State Machine.

#### 4.3. DRBAC operation

The operation of DRBAC is illustrated using the example presented in Section 3. In this example, when Professor  $B$  logs on the system in her office with a PDA, the central authority assigns her a subset of roles, for example, *Professor*, *Lecturer* and *Faculty*, based on her credentials. Then the central authority also sets up an access control agent on her PDA, which maintains the role state machine. Events issued by the context agent will trigger transitions between the roles in the role state machine. Now, consider a security policy that defines  $B$ 's active role as *Professor* when she is in the office (see figure 5, where the dashed circle is the active role), and defines the transition as: *Change role from Professor to Faculty when professor B leaves her office*.

When professor  $B$  accesses the resource in her office, the active role *Professor* is used. The resource maintains the permission state machines as shown in figure 6. The figure shows that each of the roles, *Professor*, *Faculty* and *Lecturer*, have their own permission state machines. The dashed circle represents the current active permission for each role. The *null* means the role does not have permission to access the resource. Similar to the role state machine, the context agent at the resource will trigger transitions in the permission state machine. In this example, we assume that the active permis-

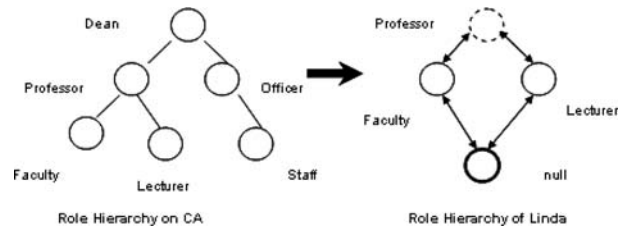


Figure 5. Role hierarchy for the smart building.

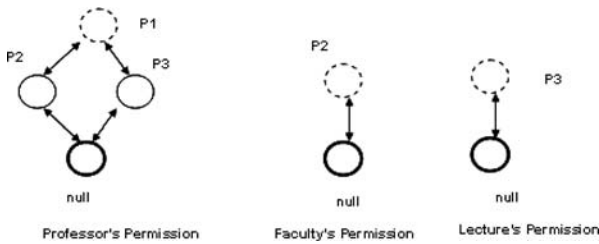


Figure 6. Permission hierarchy for the resource.

sion of the role *professor* is  $P_1$  while the system load of the resource is low.  $P_1$  means both read and write privilege. The security policy for the resource may define a permission transition for role *professor* as: *Transit permission from  $P_1$  to  $P_2$  when the system load is high*. The permission  $P_2$  means only read privilege.

Based on the situations defined above, we can describe some scenarios to illustrate dynamic access control.

- When professor *B* moves out of her office, the context agent will send an event to the access control agent on her PDA. This event will trigger a transition in the role state machine, changing her active role to *Faculty*. As a result, professor *B* will not be able to write to resource once she leaves her office as role *Faculty* only has the permission  $P_2$  or null.
- When professor *B* accesses the resource in her office, her active role is *professor*, which has both read and write privilege on the resource as long as the system load of the resource is low. If the system load becomes critically high, the resource permission state machine will change the active permission for professor *B*'s role *professor* to  $P_2$  and she will lose the privilege to write the resource.

From the scenarios described above, we see that DRBAC can enhance the security of the pervasive applications. The DRBAC mechanism implemented in this application guarantees that professor *B*'s privilege to access the resource will be changed dynamically when the context changes. Using context information to change the user's privileges prevents resources from being incorrectly used.

## 5. SESAME/DRBAC prototype implementation

A prototype of SESAME and the DRBAC model has been implemented as part of the Discover [2, 18] computational collaboratory. Discover is a Grid-based computational collaboratory that enables geographically distributed scientists and engineers to collaboratively access, monitor, and control distributed applications, services, resources and data on the Grid using pervasive portal. Key components of the Discover collaboratory include:

- *Discover Collaborative Portals* [18] that provide users with pervasive and collaborative access to Grid applications, services and resources. Using these portals, users

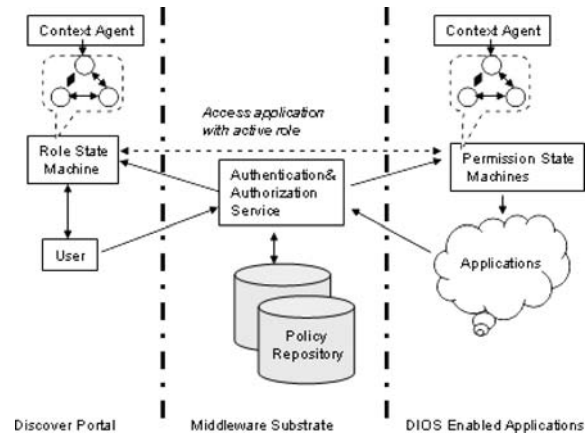


Figure 7. Dynamic access control in discover.

can discover and allocate resources, configure and launch applications and services, and monitor, interact with, and steer their execution.

- *Discover Middleware Substrate* [2, 12] that enables global collaborative access to multiple, geographically distributed instances of the Discover computational collaboratory, and provides interoperability between Discover and external Grid services such as those provided by Globus [15].
- *DIOS Interactive Object Framework (DIOS)* [13] that enables the runtime monitoring, interaction and computational steering of Grid applications and services. DIOS enables application objects to be enhanced with sensors and actuators so that they can be interrogated and controlled.

An overview of the integration of SESAME and DRBAC with Discover is presented in figure 7. SESAME ensures the users can access, monitor and steer Grid resources/applications/services only if they have appropriate privileges and capabilities. As Discover portals are pervasive and the Grid environment is dynamic, this requires dynamic context aware access management. Note that authentication services are provided by GSI [6] in our prototype implementation.

In our implementation, users entering the Discover collaboratory using the portal are assigned a set of roles when they log in. A *Role State Machine* is then locally set up for each user, which dynamically adjusts the active role based on events from the local context agent. Similarly, the *Permission State Machines* are set up at the application (or service/resource) for each role that will access it. The *Permission State Machines* similarly adjust the active permissions based on events from the local context agent. The context agents are authorized by the central authority using GSI delegation mechanisms. The access control policy is stored in the policy repository, which is maintained by an *Authentication & Authorization Service* within *Discover Middleware Substrate*. Policies are specified in XML and define role/permission assignments and transitions as illustrated in figure 8.

```

<ROLE_TRANSITION>
  <POLICY>
    <SUBJECTID>gszhang</SUBJECTID>
    <BEGIN_ROLE>Super User</BEGIN_ROLE>
    <EVENT>Unsecure Link</EVENT>
    <END_ROLE>General User</END_ROLE>
  </POLICY>
</ROLE_TRANSITION>

```

Figure 8. Sample RoleTransition policy in XML.

Policies defined for our implementation include *UserPolicy*, *RoleHierarchyPolicy*, *RoleAssignmentPolicy*, *PermissionAssignmentPolicy*, *EventPolicy*, *RoleTransitionPolicy* and *PermissionTransitionPolicy*.

In our prototype implementation, we assume that a security administrator will guarantee the correctness of a policy for a object or subject—i.e. SESAME sets up the *Role State Machines* and *Permission State Machines* without considering checking them for errors or conflicts. There are no inherent constraints on the number of roles and permissions, or on the relationships between the roles or permissions. To illustrate our implementation, consider a simple example with a single user with three roles and a Grid resource with three permissions, as shown in Table 1 and 2 respectively. The role and permission hierarchies for this example are shown in figure 9.

We consider two types of context information in our implementation: (1) Object context such as a user's location, time, local resource state and link state, and (2) Subject context, such as the current load, availability, connectivity for a resource. Context agents build on existing Grid middleware services. For example object context can be collected using the Context Toolkit [1] and subject context can be obtained using NWS [17].

Table 1  
Permission assignments for the example.

Role	Permissions
Super user	$P_1, P_2, P_3$
Basic user	$P_2, P_3$
Guest	$P_3$

Table 2  
Permission definition for the example.

Permission	Privileges
$P_1$	Steer Object, View Object, Basic
$P_2$	View Object, Basic
$P_3$	Basic

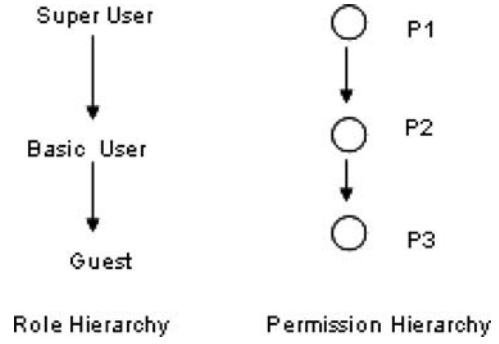


Figure 9. Role and permission hierarchies for the example.

### 5.1. SESAME/DRBAC operation

The operation of the prototype is illustrated using a set of simple scenarios. These scenarios, although somewhat contrived, demonstrate the effectiveness and utility of the DRBAC model for Grid applications. For each of these scenarios, consider a user (say  $N$ ) equipped with a mobile devices such as a PDA, and involved in collaboration scientific investigation using Discover. Assume that the user's environment is part of the pervasive Grid environment with appropriate middleware services.

Assume that user  $N$  logs into the system using her PDA. Based on her credentials, the *Authentication & Authorization service* assigns her a set of roles. The *Authority Service* also sets up an access control agent on her PDA, which maintains the role state machine. A DRBAC policy defined to select an appropriate role based on the level of security of her wireless connection, i.e. her active role is *Super User* while the network is secure (e.g. in her laboratory or office) and is *Basic User* if it is insecure. The corresponding *EventPolicy* and *RoleTransitionPolicy* may be defined as follow:

- *EventPolicy*—Generate event *insecure* when  $N$ 's link has no encryption.
- *RoleTransitionPolicy*—Transit role from *Super User* to *Basic User* when event *insecure* is generated.

A corresponding permission state machine is maintained on the application side as shown in figure 10. As seen in the figure each role has its own permission state machine. The dashed circle represents the current active permission for each role. A DRBAC policy is defined so that the active permission of the role *Super User* is  $P_1$  while load is low and  $P_2$  when the system load increases above some threshold, as there is a possibility that the application may get corrupted. The corresponding *EventPolicy* and *PermissionTransitionPolicy* may be defined as follow:

- *EventPolicy*—Generate event *highload* when load increases above *Threshold*.
- *PermissionTransitionPolicy*—Transit permission from  $P_1$  to  $P_2$  when event *highload* is generated.

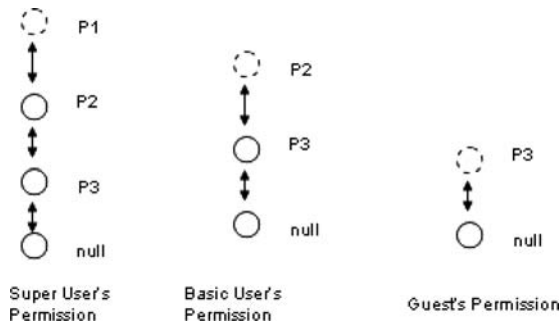


Figure 10. Permission hierarchy for the application

Based on the policies defined above, the following scenarios illustrate the operation of the SESAME DRBAC model.

- When user *N* moves out of her laboratory, the context agent will detect (using middleware context services) that the wireless network no longer has the level of encryption required and will generate the *insecure* event. This event will trigger a transition in the role state machine and downgrade her active role to *Basic user*. As a result of this transition, *N* will not be able to control and steer applications as she did while in her laboratory. When she reaches her office where the network is once again secure, the agent will detect this and will once again make *Super User* the active role.
- While in her office, *N*'s active role is *Super User* and she can monitor, interact with and steer applications under normal circumstances (load at the application server is low). However if the load on the application server increases as more users join the session, the local agent generates the *highload* event, which triggers a transition in

the permission state machine and change from *P<sub>1</sub>* to *P<sub>2</sub>*. As a result *Super User* will no longer be able to steer the application.

A screen dump from the *Discover Portal* during these scenarios is illustrated in figure 11. As shown in this figure, due to the transitions, the portal displays “You don’t have the permission to access ...”. Note that for these scenarios and the experiments presented in the following section, context information was simulated.

In our current implementation of the DRBAC model, the active role of the user and the active permission of the role change independently. As a result, it is possible that even though the active role of user has been changed to match the current context, the user has certain permission(s) based on the previous role. We are currently addressing this potential consistency issue.

### 6. Experimental evaluation

We use the prototype implementation of SESAME in Discover to measure the overheads of the DRBAC model. The experiments were conducted on two PC using PII-200 MHZ processors, running Windows NT 4.0, and one PC using PIII-500 MHZ processor, running RedHat Linux 7.2. The machines were connected by a 100 Mb Ethernet switch. The *Discover Middleware* was installed on the machines running Windows NT 4.0, while the *Application* was installed on the machine running RedHat Linux 7.2. The *Discover portal* ran on the other machine running Windows NT 4.0. The following factors affect overhead of the DRBAC model.

- The number of roles assigned to the object.

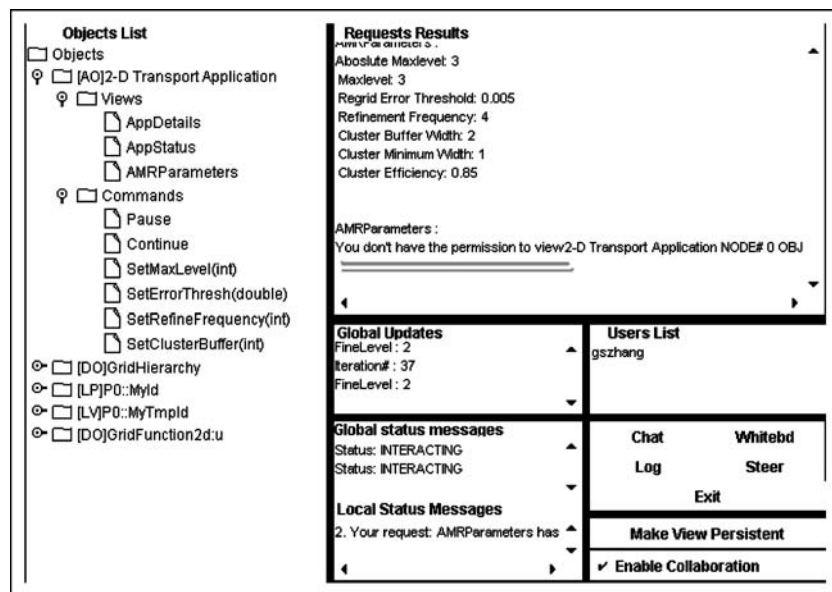


Figure 11. Dynamic access control in discover

Table 3  
Interaction time in ms. for different context event frequencies.

Event frequency	Time (ms)
–	2300
1 min	4732
2 min	4403
3 min	4102
4 min	3482
5 min	3104

- The frequency of the events (generated by the context agent at the object) that trigger transitions in the role state machine.
- The number of permissions assigned to each role.
- The frequency of the events (generated by the context agent at the subject) that trigger transitions in the permission state machine.

In the first set of experiments, we assigned each user 5 roles, and the role with highest privileges had 5 permissions. The events that triggered transitions in the role state machine were generated at different time interval. The times required to generate a request at the *Discover Portal* and get a response from the *Applications*, i.e. the interaction times, for different event frequencies are listed in Table 3. The first row is for the case without DRBAC.

In the second set of experiments, we randomly generate events to trigger transitions in the role state machine and vary the number of roles assigned. The role with the highest privileges is still assigned 5 permissions. Table 4 shows the interaction times for different number of roles.

In the last set of experiments, the user had a state machine with 5 roles and the role with the highest privileges was set as the active role. Events were randomly generated at the application server to trigger transitions in the permission state machine. The number of permissions assigned to the active role was varied. The interaction times for different number of permissions are listed in Table 5.

These preliminary results show that in general the overheads of the DRBAC implementation are reasonable. The primary overheads were due to the event generated by the context agent—the higher the frequency, the larger was the overhead. The context agent can be implemented as an independent thread and as a result, the transition overheads at the object and subject are not significant.

Table 4  
Interaction time in ms. for different number of roles.

Number of roles	Time (ms.)
–	2300
5	2520
6	2608
7	2804
8	2920
9	3004

Table 5  
Interaction time in ms. for different number of permissions.

Number of permissions	Time (ms)
–	2300
5	2500
6	2602
7	2698
8	2804
9	2912

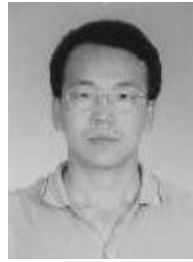
## 7. Conclusion

In this paper, we presented the Dynamic Role Based Access Control (DRBAC) model that provides context aware access control for pervasive applications. DRBAC extends the role based access control (RBAC) model and dynamically adjusts *Role Assignments* and *Permission Assignments* based on context information. The operation of the model was illustrated using a sample application scenario. Compared to traditional access control mechanisms, the DRBAC model can provide improved security for pervasive applications. However, access control alone is not sufficient and DRBAC must be combined with feasible authentication mechanisms to secure pervasive applications in the real world.

## References

- [1] G.D. Abowd and A.K. Dey, The context toolkit: Aiding the development of context-aware applications, in: *Human Factors in Computing Systems: CHI 99*, ACM Press, eds, Pittsburgh, PA, USA, (May 1999) pp. 434–441.
- [2] V. Bhat and M. Parashar, A middleware substrate for integrating services on the grid, Technical Report Technical Report Number TR-268, ICenter for Advanced Information Processing, Rutgers University, November 2002.
- [3] S. Gavrilu, D.R. Kuhn, D.F. Ferraiolo, R. Sandhu and R. Chandramouli, Proposed nist standard for role-based access control, *ACM Transactions on Information and System Security* 4(3) (2001) 224–274.
- [4] National Science Foundation. National Ecological Observatory Network Project Web Site, <http://www.nsf.gov/bio/neon/start.htm>.
- [5] L. Giuri and P. Iglu, Role templates for content-based access control, in: *Proceedings of the Second ACM Workshop on Role Based Access Control*, Virginia, USA (1997).
- [6] G. Tsudik, S. Tuecke, I. Foster and C. Kesselman, A security architecture for computational grids, in: *5th ACM Conference on Computer and Communications Security Conference*, San Francisco, CA, USA (1998) pp. 88–92.
- [7] R. Campbell, J.AI-Muhtadi, A. Ranganathan and M.D. Mickunas, A flexible, privacy-preserving authentication framework for ubiquitous computing environments, in: *International Workshop on Smart Appliances and Wearable Computing*, Vienna, Austria (2002).
- [8] K. Beznosov, J. Barkley and J. Uppal, Supporting relationships in access control using role based access control, 1999.
- [9] J. Elson, H. Wang, D. Maniezzo, R.E. Hudson, K. Yao, J.C. Chen, L. Yip and D. Estrin, Coherent acoustic array processing and localization on wireless sensor network, *IEEE Proceedings* 91(8), August (2003).
- [10] M.J. Moyer, M.J. Covington and M. Ahamad, Generalized role-based access control for securing future applications, in: *23rd National Information Systems Security Conference. (NISSC 2000)*, Baltimore, Md, USA (October 2000).

- [11] S. Srinivasan, A. Dey, M. Ahamad, M.J. Covington, W. Long and G. Abowd, Securing context-aware applications using environment roles (May 2001).
- [12] V. Mann and M. Parashar, Engineering an interoperable computational collaboratory on the grid, Special Issue on Grid Computing Environments, *Concurrency and Computation: Practice and Experience* 14(13/15) (2002) 1569–1593.
- [13] R. Muralidhar and M. Parashar, A distributed object infrastructure for interaction and steering, in: *Concurrency and Computation: Practice and Experience*, to appear.
- [14] Massachusetts Institute of Technology. The IntelligentRoom Research Project Web Site, <http://www.ai.mit.edu/projects/iroom/index.shtml>.
- [15] Globus Project. Globus Project Web Site, 2003. <http://www.globus.org/>.
- [16] H. Feinstein, R. Sandhu, E. Coyne and C. Youman, Role-based access control models, *IEEE Computer*, 29(2) (1996) 38–47.
- [17] Network Weather Service. University of California, Santa Barbara, Research Project Web Site, 2003. <http://nws.cs.ucsb.edu/>.
- [18] R. Muralidhar, V. Mann, V. Matossian and M. Parashar, Discover: An environment for web-based interaction and steering of high-performance scientific applications, *Concurrency and Computation: Practice and Experience* 13(8/9) (2001) 737–754.
- [19] T.Y.C. Woo and Simon S. Lam, Designing a distributed authorization service, in: *Proceedings of IEEE INFOCOM*, 1998.



**Guangsen Zhang** is Ph.D. student in the Department of Electrical and Computer Engineering at Rutgers University. He received his MS from Rutgers University. His research interests include parallel & distributed computing, distributed system security.

E-mail: [gszhang@caip.rutgers.edu](mailto:gszhang@caip.rutgers.edu)



**Manish Parashar** is an Associate Professor in the Department of Electrical and Computer Engineering at Rutgers University. His research interests include autonomic computing, parallel & distributed computing, scientific computing, and software engineering.

E-mail: [parashar@caip.rutgers.edu](mailto:parashar@caip.rutgers.edu)