Role Based Access Control in Enterprise Application – Security Administration and User Management

Vinith Bindiganavale and Dr. Jinsong Ouyang, *Member, IEEE*

***Abstract*—One of the most challenging problems in managing large web-applications is the complexity of security administration and user-profile management. Role Based Access Control (RBAC) has become the predominant model for advanced access control because it reduces the complexity and cost of administration. Under RBAC, security administration is greatly simplified by using roles, hierarchies and privileges, and user management is uncomplicated by using LDAP API specification within the J2EE application. System administrators create roles according to the job functions performed in an organization, grant permissions to those roles, and then assign users to the roles on the basis of their specific job responsibilities and qualifications.**

This article introduces RBAC in a typical J2EE enterprise application and provides architectural details, along with security administration and user-profile management for RBAC. Netegrity[1] SiteMinder provides the RBAC foundation and J2EE framework serves as the reference model for administration in the application. In this article, more emphasis is given to the design and implementation of a custom RBAC-model, and the possibilities of optimization of this model.

I. Introduction

Access control has existed as a concept for as long as humans have had assets worth protecting. It has been so subtly stated that the perception of access control has been around for a very long time, both intentionally and inadvertently. The definition and modeling of access control stem from seminal papers of the early 1970s, the early standardization efforts of the 1980s, and the emergence of RBAC that began in the early 1990s, and it continues to this day.

The earliest work in defining a formal, mathematical description of access control is that of Lampson, who introduced the formal notion of *subject* and *object,* and an *access matrix*. An access matrix is a simple conceptual representation in which the (*i*,*j*) entry in the matrix specifies the rights that subject *i* has to object *j*, as shown in the sample table below.

Dr. Jinsong Ouyang is a professor with California State University, Sacramento, CA 95819 USA (phone: xxx-xxx-xxxx; fax: xxx-xxx-xxxx; e-mail: ouyangj@ecs.csus.edu). All questions should be directed to Prof. Ouyang.

Vinith Bindiganavale was a graduate student in California State University, Sacramento, CA 95819 USA. He graduated from the Department of Computer Science in spring 2006 (e-mail: bindigav@ecs.csus.edu or bsvinith@hotmail.com).

[1]The company *Netegrity*® was previously related to Computer Associates® (CA), and currently associated with eTrust®.

TABLE I
SAMPLE ACCESS MATRIX

	Initiate Case	Record Payment	Create User
Clerk	R	R, W	R
Judge	R, W	R	R
Admin			R, W

R: Read
W: Write

Access control is critical to preserve the three most important information security risks:

- *Confidentiality* refers to the assurance that information is kept secure and private.
- *Integrity* refers to the concept of keeping the information authentic and complete.
- *Availability* refers to the notion that information is available for use when needed.

Among other wide categories of essential computer security solutions which are necessary, access control is the most dominant among them and also the most visible.

A. Access Control

In the broadest sense, apart from controlling the access to a computer, access control also refers to the general way of inhibiting access to web resources, including restrictions based on time of day, IP/MAC address of the client machine, domain of the URI, type of encryption the client can support, the frequency of user authentication in a day, the possession of types of hardware/software tokens, or responsibilities of employees in the firm.

B. Role based access control

This paper is based on an advanced access control mechanism that uses job responsibilities (or roles) of employees in the organization. With RBAC, access decisions are based on the roles that individual users have as part of an organization. Users are assigned roles (such as clerk, judge, cashier, supervisor, administrator), and roles have permission-rights on certain resources (such as top-menus, left-menus, web-pages, buttons, database views). Access rights are grouped by role name, and the use of resources is restricted to individuals authorized to assume the associated role.

The administration is simplified because the roles defined for an organization tend to be relatively static entities, whereas the users who are assigned to these roles change frequently. Similarly, the set of permissions associated with a role are expected to be stable, whereas access to the resources may be dynamic. This mapping can be represented as shown in Fig 1 below.

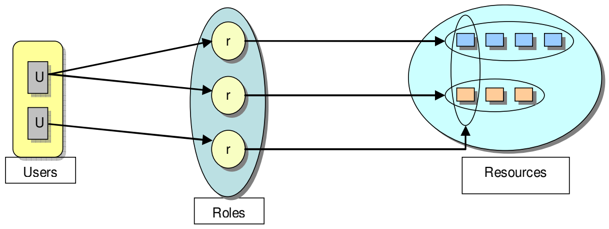

Fig 1. Core RBAC elements – users, roles and resources. A theoretical complexity of this scenario is:

$$O(|s| \times |r|) + O(|r| \times |o|)$$

where, s: users r: roles o: objects

C. *Gap Analysis*

The RBAC security model is both abstract and general. It is abstract, because properties that are not relevant to security are not included, and it is general because many designs could be considered valid interpretations of the model. Thus, the model is usable as a basis for the design of a variety of IT systems. Because of the wide range of possible RBAC deployments, different RBAC features apply to different environments based on their scope of control and risk profile. This un-standardization leads to a gap that exists between the theoretical concepts of RBAC and its actual implementation in an enterprise application.

This article tries to define a framework of RBAC that could be reused in other applications[1] with minimal changes. Effort is also made to optimize the custom model, in terms of design/architecture, product versions, queries and their execution, and deployment.

D. *Goal of Document*

The goal of this article is to design an easy-to-use, pluggable access control mechanism that could be easily integrated into an enterprise J2EE application. The objective of having a RBAC-based security layer in the application are – centralized management, delegated administration, robust authentication/authorization, security abstraction from the application-domain, flexibility for vertical/horizontal growth, and optimize access speed. This article explores various facets of the security features implemented, including infrastructure specification (hardware/software), architectural overview, design aspects, functional details, and business processes.

E. *Layout*

This article is organized into eight sections. Each of these sections provides brief details of its purpose in the design, and explains the RBAC model.

Section I (*Introduction*) provides a starting point to understand access control, RBAC features, and the purpose of this article/design.

Section II (*RBAC Specification*) goes into details of RBAC model proposed in this article, and introduces the components used.

[1]Note that this model is inclined towards J2EE web-applications, but it could also be used in non-J2EE and stand-alone applications too.

Section III (*RBAC Model*) provides a brief overview about requirements and principles of the web-application. It lists the various security-levels integrated into the system, and also the user-profile user-interfaces proposed.

Section IV (*Design & Implementation*) is the most descriptive section of this article. It provides details of each component involved in the system to realize RBAC, and wherever applicable, provides optimization details.

Section V (*RBAC Operation*) provides the complete flow and mechanism of the system designed.

Section VI (*Results*) gives the analysis and outcome of the tuning activity performed on the Directory Server.

Section VII (*Conclusion*) summarizes the purposes and achievements of this article, and briefly mentions the status of RBAC today.

II. RBAC Specification

In this section, an overview of various softwares used for the design will be provided. The software specification needed for security implementation, both in the security-tier (Netegrity section) and the business-tier (Java code) will be explained briefly.

After a grand period of analysis and study, a technology-stack diagram was produced that documents the selected software components from a high level. This technology stack summarizes a Java-based solution focusing on *best-of-breed* products from Microsoft, Sun, BEA, Oracle, and Netegrity. Fig 2 below illustrates the overall software technologies used in the design, along with an introduction to the logical architecture. Each of the specification is briefly explained in terms of a functional description, along with their purpose in the model.

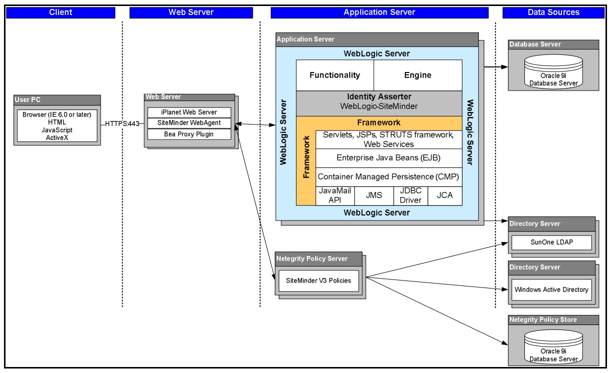

Fig 2. Proposed Logical Specification for RBAC model. This design utilizes a n-tier, multi-pattern J2EE model, with four significant layers – Browser, Web Server, Application Server, and Data-source.

A. *Web Server*

SunOne (formerly iPlanet) Web Server is used as the primary JSP/Servlet container for the application; it is one of the four Web Servers supported by the BEA WebLogic platform. There are two other software components that are implanted into the Web Server – WebLogic Proxy Plug-in and the SiteMinder WebAgent. The *BEA WebLogic proxy plug-in* is a software component that brokers the communications between the SunOne Web Server and the WebLogic

Application Server. The *SiteMinder WebAgent* is a software component that enforces the security administration functions of the Web Server, by controlling access to protected resources in the application. This WebAgent works with Netegrity Policy Server to authenticate and authorize users.

B. *Policy Server*

The heart of the RBAC security functionality is the Netegrity SiteMinder[2]. It is a security management foundation for enterprise Web applications with a centralized security infrastructure for managing user authentication and access, and provides a platform for access control that operates in conjunction with:

- Policy Store (database)
- Directory Server (LDAP)
- Active Directory (AD)

AD is used for user authentication (or user-password validation), LDAP [3] for authorization (or user-role membership), and the database is used as the repository for storing the data of the Policy Server (policies, system objects, configuration details, rules, responses, etc.).

By assigning access control policies to specific resources, a fine grained authorization over what a user has access to can be controlled.

C. *Application Server*

As mentioned before, BEA WebLogic Server is used as the Application Server, as it is compatible with the iPlanet Web Server and Netegrity Policy Server. It provides a complete set of services for J2EE development platform, used for developing and deploying this multi-tier distributed enterprise application. WebLogic Server is the place where the actual bulk of the module-code and the framework-code are deployed. This interacts with the database server, Policy Server and other servers to complete a unit of work.

One of the key components of the Application Server is the *Identity Assertion Provider* (or Identity Asserter or AppAgent). The AppAgent used in the design is the Netegrity SiteMinder Agent for BEA WebLogic. Once a user is successfully authorized by the WebAgent through the Policy Server, it passes the authenticated user information in the form of a token to the AppAgent through the WebLogic Security framework in the WebLogic Server domain. This AppAgent validates the token through a direct call to LDAP server and obtains authenticated user information, enabling WebLogic to trust the requests and not prompt users for re-authentication.

III. RBAC MODEL

This article has a list of pre-defined security requirements to fulfill the RBAC functionality of the application. These requirements can be categorized into functional, infrastructural and administrative needs, or unified into a single term – *Security Principles* of the project. These principles serve as general guidelines for the design process of all aspects of the application security.

Security principles have been developed based on a combination of industry standards, best practices and real world experience in the implementation of similar application-security features. The RBAC model is two-folds:

- Security-levels
- UI functionalities

A. *Security Levels for RBAC*

The proposed design provides a comprehensive access control solution, in terms of different levels of security by using SiteMinder and WebLogic security framework. The security levels integrated into this system are: Resource, Menu, and Field.

Resource Level Security: This is the security level used to protect the web resource from the hackers who try to access the application by either copying the URL or from the bookmarks or through any other means. This is implemented by listing all the resources in the *Rules* of Policy Server.

When the user clicks the logoff button to terminate the current activity, the WebAgent triggers the immediate suspension of the user's SiteMinder session and Java session. The same behavior is replicated during session time-out and in the scenario where the user closes the web browser.

Menu Level Security: Here, users see only those menu items (both on top and left) of the application that they have access to, based on role assignments. Menu names are stored in LDAP and tied to the roles for a user; a menu policy is created for the protected URL and is configured to send the resource-id for the configured menu items via SiteMinder responses based on the user's role membership.

Field Level Security: Here, instead of selectively rendering menus, widgets (or fields) in a web-page are painted based on roles. Every field on a JSP page has a field-id and is configured as part of a *Response* in SiteMinder; these *Responses* are tied to a URL-*Policy* and are fired based on role-assignment.

B. *User Interface for RBAC*

Apart from the RBAC functionality that is configured and setup in SiteMinder, the actual user-profile management is performed as part of the application-feature. User profiles are administered through the application Admin-screens. These screens allow administrators to maintain user creation and role assignments. These are:

- Create user
- Search user (with different criterion)
- Update profile
- Role assignment

IV. DESIGN & IMPLEMENTATION

The custom RBAC model designed in the article is explained in this section. Here, the security components listed before have been depicted and their physical layout described briefly. These components include: iPlanet Web Server, SunOne Directory Server, Microsoft Active Directory and the SiteMinder Policy Server.

[2] SiteMinder is a product from Netegrity (previously Computer Associates, and now eTrust). In this article, SiteMinder and Policy Server are used interchangeably.

[3] Note that in the rest of the article, Directory Server will also be referred to as LDAP, for simplicity.

A. *iPlanet Web Server*

This Web Server provides high performance, reliability, scalability, and manageability for any size enterprise. In addition, it offers the following support that makes it very suitable for this design: Delegated administration, cluster management, LDAP support, client/server security, access control, and Java Servlet/JSP support.

Apart from the Admin Server instance in the console, there are two other instances:

- Actual application Web Server instance
- Policy Server Admin console instance

A second instance is recommended for performance and maintenance issues. There are two configuration files on the Web Server which are customized to accommodate SiteMinder: *magnus.conf* and *obj.conf.*

B. *Microsoft Active Directory*

In the model, the AD is a LDAP compliant Directory Serer that acts as a static repository of user-ids and passwords, with a very few specialized services provided. All users who access this application are required to be authenticated by the AD. SiteMinder connects to AD during login to perform a lookup of the current user's entered credentials, and ensure that the user-id exists in the directory and the password matches for the user.

Since AD is used for simple lookup, all users are stored directly under the *Users* rootDN:

```
dn=CN=Users,DC=application,DC=com
```

This makes querying simple and faster, although not efficient for very large number of users. As mentioned before, when configuring AD as the authenticating directory in SiteMinder, the optimized query is:

```
(& (sAMAccountName=%u) (objectClass=user))
```

Here, *sAMAccountName* is the attribute used to identify a user uniquely, and *(objectClass=user)* is the filter criteria.

C. *SunOne Directory Server*

Directory Server is the cornerstone for building a centralized and distributed data repository. In this design, LDAP acts as the heart of RBAC security for satisfying the access control mechanism for the application and stores all the obligatory information needed by the Policy Server to map users to roles. The information stored in LDAP is the user-profile data and role-profile data, in accordance with the LDAP structure (shown in Fig 3).

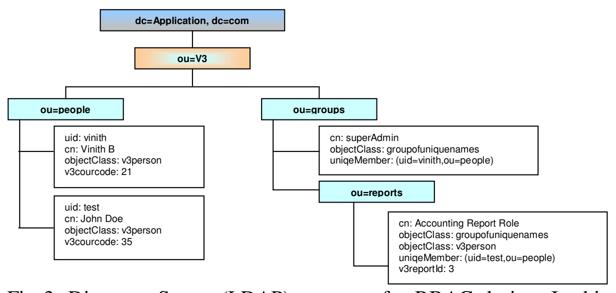

Fig 3. Directory Server (LDAP) structure for RBAC design. In this schema, user-names are put under *ou=people* and all role-names under *ou=groups* or a custom sub-group (eg. *ou=reports*).

For efficient lookup, the schema consists of two distinct organization units (*ou*): *people* and *groups*. Each entity in *people* is a user-profile that stores the user information and each entity in *groups* has role information that stores RBAC data; users are associated to roles using a *uniqueMember* mapping.

The other directory that is used internally with Policy Server during configuration, is the LDAP (used for authorization of users). The *userDN* lookup query is:

```
    (& (uid=%u) (objectClass=person))
```

and the *rootDN* is:

```
dn=ou=people,ou=V3,dc=Application,dc=com
```

Apart from the standard SunOne provided schema with the regular set of *objectClasses* and *attributes*, a few custom ones (Table II) have been added. These are needed to pass access control parameters between SiteMinder and the Java application.

TABLE II
ALL LDAP *objectClasses* AND *attributes* IN DESIGN

objectClass	Attribute	Description
inetUser	inetUserStatus	Takes binary values (Yes/No) – to indicate user is enabled/disabled for authorization
person	cn	Common Name – used as a naming attribute to refer to a user's full name; used in a group to locate a role-name
	sn	Surname – used to refer to a user's last name
	l	Location – used as an attribute to indicate a user's court location (for example)
v3person	v3topMenu	All Top Menus are stored in this attribute
	v3leftmenu	All Left Menus are stored in this attribute
	v3courtCode	Refers to a user location's court
	v3reportId	Stores report-ids for RBAC control of report generation
groupofuniqenames	uniqueMember	This multi-valued attribute takes the values of *uid*s of users in *ou=people*

D. *Netegrity Policy Server*

SiteMinder enables administrators to assign authentication schemes, define and manage authorization privileges to specific resources, and create rules and policies to implement these authorization permissions. A SiteMinder environment mainly consists of two core components:

- Policy Server – provides policy management, authentication, authorization, and accounting services.
- SiteMinder Agents – enables SiteMinder to manage access to the web application and contents according to predefined security policies. There are two types of Agents used in this application:
    - *WebAgent*: plugs into the Web Server
    - *AppAgent*: on the Application Server

There are two configurable elements: Host Configuration Object (HCO) and Agent Configuration Object (ACO), which needs to be customized for performance.

In the design, the SiteMinder Policy Server is the nucleus of RBAC security for the application, and provides Authentication and Authorization features. Apart from connecting to AD and LDAP, the Policy Server stores *Policies* for performing access control. Using these Policies, objects that identify users, resources and actions associated with the resources, are linked (bound) together. This is shown in Fig 4 below. For this J2EE application, *Rules* are mostly URIs for accessing screens and *Responses* are action to be performed on the *Rules*.

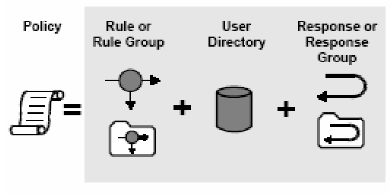

Fig 4. Components of a *Policy*. It consists of user-directories, Rules (or Rule-Groups) and Responses (or Response-Groups).

In the proposed design, a custom *Policy Domain* is created that has realms with separate authentication-schemes to restrict access based on context-paths (eg. */acct/**, */kiosk/**). When a user's browser attempts to access the resource, the *Rule* fires and the *Policy* containing the *Rule* determines whether or not the user can view the selected resource, and any further action to take using a *Response*. Examples of denied URLs:

/v3/accounting/nonexistent_url.do – a non-existent URL

*/v3/admin/add_**U**ser.do* – a mal-formed URL

/v3/case/confidential_cases.do – URL not assigned to Accounting Inquiry-Clerk Access

Responses can contain attributes for HTTP header variables, cookie variables, and URLs for redirections. They can be used by JSP/Servlets to display customized content, change SiteMinder settings, or redirect users to different resources, to be used as privileges or entitlements for fine-grained access control. In the design, the name/value pair is used to send field-level data. That is, the custom JSTL tag-id (*fieldId*) of buttons, hyperlinks, checkboxes, etc., from a JSP page, is sent as responses from the Policy Server to the J2EE application, based on policies (roles and users). Using these field-ids, the widgets could be selectively rendered.

When one of the users specified in a *Policy* attempts to access a resource identified in one of the Policy's *Rules*, the Policy Server uses the information contained in the *Response* to resolve access to the resource, and if any personalization should take place.

E. BEA Application Server

The BEA WebLogic Application Server operates in the middle-tier of this multi-tier J2EE application architecture. The custom built J2EE applications uses a hardened *framework* that promotes reuse, mitigates risk and reduces development time. This custom developed framework unites familiar concepts such as display fields, application events, component hierarchies, and a page-centric development approach, with a state-of-the-art design based on the Model-View-Controller and Service-to-Workers patterns.

One of the components of Application Server that is intrinsically involved in access control is the Identity Asserter (or AppAgent). It is used to provide the access control solution to BEA WebLogic to leverage the existing BEA's Security Service Provider Interface (SSPI). This process is referred to as *Perimeter Authentication*. This method is used to provide single sign-on capabilities for the application, and trust requests serviced by SiteMinder-protected proxy server. WebLogic provides the infrastructure for external security vendors (like SiteMinder) to implement access control, through AppAgent. This is done by editing the *web.xml* to access LDAP, and setting the authentication-method to *CLIENT-CERT*.

The application-client consists of Java Servlets, JavaServer Pages and JavaScript functions, which are solely responsibile for displaying the model data. The JSP files contain Java bean components, custom tags, JSTL tiles and Struts objects; these files contain code (Display Framework) related to the display of model data and the repetitive HTML rendering is handled by tags or tiles, whenever possible. The Display Framework (shown in Fig 5) provides the basis of operation for two main display mechanisms: Menu rendering and Tab navigation.

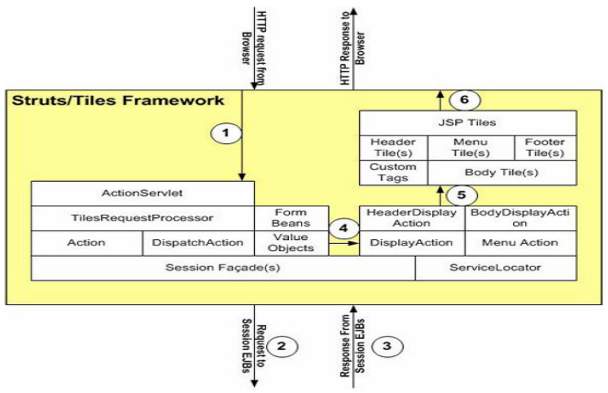

Fig 5. Display Framework for menu rendering. The framework shows components that are needed to display menu/tabs on the screen.

A HTTP request arrives at the ActionServlet and the GET/POST operations are forwarded to a TilesRequestProcessor. The request parameters are mapped to the request in the Struts configuration file. In the return trip, the Action class receives the response from Session Bean and returns an ActionForward object for 'success' mapping. The RequestProcessor locates the tiles definition returned and forwards the response to the tiles-JSP. Each of the tiles-JSP is composed of many individual JSP fragments (tiles), example: header, menu, footer and body. The retrieved data is forwarded to each of the JSP fragments to be rendered.

The tiles-configuration file for rendering top-menus is shown below:

```
<component-definitions>
  <definition name="topnavDef"
path="/WEB-INF/topnav/topnav.jsp">
    <putList name='tabList'>
      <item name="Case" index=0 link='/case/display_search.do'/>
      <item name="Accounting" index=1 link='/acct/search.do'/>
      <item name="Calendar" index=2 link='/cal/cal_view.do'/>
      <item name="Reports" index=3 link='/reports/view_rep.do'/>
      <item name="Admin" index=4 link='/admin/admin_home.do'/>
    </putList>
  </definition>
```

The tiles-configuration file for rendering left-menus and sub-left menus is shown below:

```
<component-definitions>
  <definition name="AdmLeftDef" path="/leftnav/admleftnav.jsp">
   <putList name="leftNavList">
      <item value="Case Administration"
         link="/configureroa_list.do" index="0"/>
      <item value="Accounting Administration"
         link="/con_fee_search.do" index="1"/>
      <item value="Calendar Administration"
         link="/admin/reshome.do" index="2"/>
      <item value="User Administration"
         link ="/security/add_user.do" index="3"/>
   </putList>
   <putList name="tabFirstList">
     <item value='Create User'
      link='/sec/add_user.do' index=0 parentIndex="4"/>
     <item value='Search User'
      link='/sec/search_users.do' index=1 parentIndex="4"/>
   </putList>
  </definition>
</component-definitions>
```

F. LDAP API

All user-management operations are coded into the J2EE application, and Java API from Netscape SDK is used for communication with the LDAP. The proposed web-application extensively uses the LDAP API for user-profile management, say to, create user, update profile, assign roles, and search users across locations. The other use of the LDAP API is to render the menus based on access control, say to display, top-menus, left-menus, and report-ids.

All the LDAP queries are called from Java classes in the business-tier. An example of Java code to get menus is:

```
Map getMenus(String userId, int menuType) {
  Map mapOfTopMenus = new HashMap();
  List listOfRoles = getRoles(userId);
  LDAPConnection ld = getLDAPConnection();
  LDAPEntry findEntry = null;
  while (listOfRoles.hasNext()) {
    LDAPEntry entryObj = (LDAPEntry) listOfRoles.next();
    String finalKey = menuType + entryObj.getDN();
    mapOfTopMenus.put(finalKey,
         getAttrsValuesList(entryObj, attrName, null)) ;
  }
  closeLDAPConnection(ld);
  return mapOfTopMenus;
}
```

These queries are optimized for performance and speed. Some of these are highlighted below:

- When possible, nested queries are avoided, and a loop-structure[4] is used on the resultant data. But some times, based on experience and stress-testing, nesting is found more efficient than repetitive querying to LDAP.
- Filters are used as much as possible to return a minimum sub-set of values.
- Specific *objectClasses* and *attributes* are used in queries to reduce processing time and redundancy.
- Java Session-Beans and Helper classes are used with utmost delicacy, when implementing LDAP queries.
- A *best-of-breed* approach of J2EE patterns is used, wherever business-tier and data-tier are involved.
- A number of LDAP tuning and optimization tools are used, along with a proven Directory Server configuration.

These have been further explained in the results.

[4]There is a very fine difference in processing time between using nested LDAP queries and Java loop-structures.

V. RBAC Operation

The integration of SiteMinder with iPlanet Web Server and WebLogic Application Server provides the required RBAC functionality to the application. Fig 6 outlines various components involved, and provides a brief understanding of the security operation.

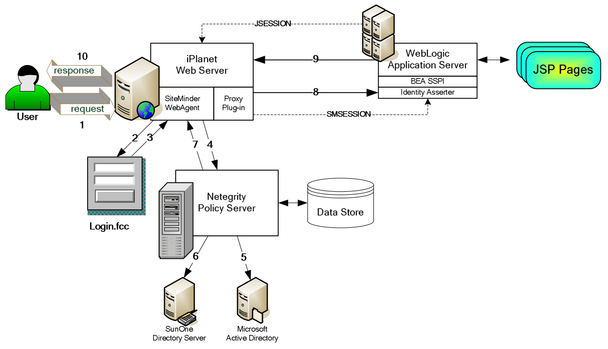

Fig 6. Complete Authentication/Authorization process. Various hardware and software components involved in RBAC operation, along with the steps involved in authentication/authorization.

The steps below explain each process of communication between the three primary components:

1. When the user makes a request to a realm with protected resource using a URI, Post data or Query-string (from the browser), the HTTP request is passed onto the Web Server to be serviced.
2. Before the BEA plug-in forwards the request to the Application Server, the WebAgent intercepts this request and determines from the Policy Server that the resource requested is contained in a realm protected by HTML forms authentication, and no legitimate session exists.
3. Based on the Authentication scheme, the WebAgent throws an authentication challenge by redirecting to the URL of credential collector file (*login.fcc*), in the browser.
4. The user fills out the custom-form (username/password) and Posts (submits) it to the credential collector (FCC) to be processed; this data is passed to the Policy Server.
5. The Policy Server does a search against the configured Active Directory using a LDAP query and authenticates if both the username and password match the existing values in AD.

6. Policy Server returns the user data to the FCC, which then creates a session cookie (*SMSESSION*) for the particular cookie-domain and passes it to the browser. After the cookie is set, the WebAgent/Policy Server starts the authorization process.
7. The policy/policies corresponding to the requested URL is fired. Policy Server does a lookup against the configured Directory Server using a LDAP query, to find the list of roles and other attributes assigned to the user in DS. It also finds the response-values corresponding to the policy/policies. All these values comprise the user-profile, and are packaged into HTTP headers/cookies and sent back to the WebAgent.
8. The WebAgent forwards these values to the BEA proxy plug-in, and finally into the Application Server. The AppAgent intercepts and performs a perimeter-authentication (explained in the next sub-section); then it creates a user-context from the user-profile data and starts a J2EE session (*JSESSION*).
9. As a last step, the requested resource (JSP-page) is displayed in the browser, with menus, fields and other role-based widgets rendered on the page, accordingly.

VI. Results

Apart from installing all the servers, configuring them, developing the application and testing it for operability, an extensive amount of testing was performed on Directory Server (LDAP). This section only covers the analysis and results of stress-test on LDAP.

A. *LDAP Stress Test*

On a SunFire v240 machine with 2×1 GHz dual-CPUs, 2 GB RAM, running Solaris 9 (latest patch level), SunOne Directory Server v5.2 was installed; the testing tool used was Mercury Interactive's LoadRunner. A market-standard benchmark was used to test virtual-users set-up in the system:

- A total of 1000 users login at the rate of 2 users every 5 seconds.
- First 500 users are super-admins (>100 roles), and the next 500 users have varying role-assignments (6-60 roles).
- These users login and wait indefinitely without performing any additional activity.

A preliminary testing revealed that the system could not support the target. High CPU usage in the LDAP directory was isolated as the limiting factor in the stress test. Also, it was determined that the most likely problem[5] was the use of static groups[6].

For upto 30 users, CPU utilization on the Sun Directory server (LDAP) is normal (<25%). Beyond 30 users, CPU utilization increases linearly with more and more users entering the system. At 300 users, CPU is 100% utilized on the LDAP server. From this point on, the time taken for a virtual user to log-in increases exponentially. At 350 users, virtual users start getting timed-out on login. This is shown in Fig 7.

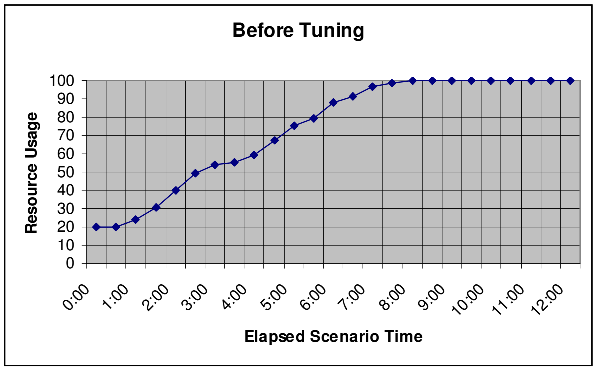

Fig 7. LoadRunner graph before LDAP tuning. CPU usage hit 100% at ~260 users, ramp-up was 2 users every 5 seconds. Note that this graph is not the actual software generated chart, but the data generated by another testing tool, *jMeter*, which is plotted.

Based on various analyses, LDAP/application log files, and LDAP provided tuning softwares (*searchrate*, *idsktune*, *idsconfig*, etc.), the following three changes were necessary:

- **Install the latest version of Java Enterprise System Directory Server 5.2.**

Directory Server v5.2 release has some issues regarding Static Group query performance even with relatively small membership (< 2000 members). This has been resolved in subsequent patches by adding a configuration parameter (nsslapd-search-tune).

- **Enable search optimization using the nsslapd-search-tune parameter.**

The nsslapd-search-tune parameter was introduced as part of the Directory Server 5.2 patch2 release. In Directory Server's *dse.ldif* file, under:

```
dn: cn=config,cn=ldbm database,cn=plugins,cn=config
```

add the attribute/value pair to the entry:

```
nsslapd-search-tune: 49
```

- **Modify LDAP query to only retrieve specific LDAP attributes instead of all attributes.**

As mentioned before, change the search-base of all LDAP queries accessed by the application to return only required attributes instead of the whole set.

A second iteration of the LoadRunner resulted in the following stress-test graph:

[5]Some preliminary testing using managed roles showed much better performance.

[6]Static Groups use *ou=groups* and user-memberships are done using *uniqueMember* attribute; Managed Roles (or Dynamic Groups) use *nsroles* attribute within *ou=people*.

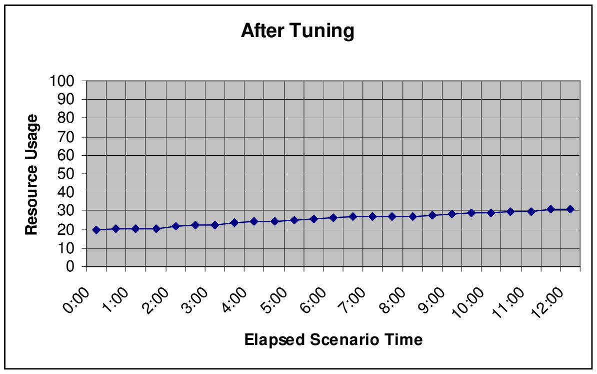

Fig 8. LoadRunner graph after LDAP tuning. A simple curve was recorded that was much below the 50% CPU usage; all 1000 users logged in successfully. Note that this graph is not the actual software generated chart, but the data generated by another testing tool, jMeter, which is plotted.

VII. Conclusion

The design in this article has implemented one of the many RBAC solutions involved in a humungous web-based, J2EE application. It explores various facets of security administration and user management, and presents a framework that systematically spans the spectrum from simple to complex implementations standards.

The application is a custom-built, multi-tier, standards-based system that has harnessed the advantages of SiteMinder for access-control and could be plugged into an enterprise application, with little challenges. It also has the benefits of integrating other servers and directories, for low-cost and high-performance implementation.

Role-Based Access-Control security is a highly dynamic area with latest developments extending the scope beyond the boundaries of an enterprise. It is becoming a major platform in many areas involving IT-security. Much remains to be done to realize the promises of RBAC.

Appendix

Acknowledgment

References

[1] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli, "*Role Base Access Control*", Artech House, Computer Security Series, 2003

[2] R. Sandhu, *"Role-Based Access Control Models"* IEEE Computer 29(2), 1996

[3] Netegrity "*SiteMinder v6.0 Complete Manual*", 2004, Available http://support.netegrity.com

[4] Sun Microsystems *"Netscape Directory SDK 4.0 for Java Programmer's Guide"*, 1999, Available http://docs.sun.com/source/816-6402-10/contents.htm

[5] BEA "*WebLogic Server and WebLogic Express 8.1*" Documentation, 2003, Available http://e-docs.bea.com/wls/docs81/

[6] I. Singh, B. Stearns, M. Johnson, Sun Microsystems "*Designing Enterprise Applications with the J2EE^{TM} Platform, Second Edition*", February, 2002

[7] NIST/ITL Bulletin, *"An Introduction to RBAC"*, December, 1995 Available http://csrc.nist.gov/rbac/NIST-ITL-RBAC-bulletin.html

[8] The Apache Foundation, "*Apache Struts Project*", July 2004, Available http://struts.apache.org/