

# Analysis of ANSI RBAC Support in EJB

Wesam Darwish and Konstantin Beznosov

Laboratory for Education and Research in Secure Systems Engineering  
[lersse.ece.ubc.ca](http://lersse.ece.ubc.ca)

Department of Electrical and Computer Engineering  
University of British Columbia  
Vancouver, Canada

Technical report LERSSE-TR-2009-XX\*

Revision: #36

Revision Date: 2009/01/21

---

\*This and other LERSSE publications can be found at <http://lersse-dl.ece.ubc.ca>

---

## Abstract

We analyze access control mechanisms of the Enterprise Java Beans (EJB) architecture and define a configuration of the EJB protection system in a more precise and less ambiguous language than the EJB 3.0 standard. Using this configuration, we suggest an algorithm that formally specifies the semantics of authorization decisions in EJB. We analyze the level of support for the American National Standard Institute's (ANSI) specification of Role-Based Access Control (RBAC) components and functional specification in EJB.

Our results indicate that the EJB specification falls short of supporting even Core ANSI RBAC. EJB extensions dependent on the operational environment are required in order to ANSI RBAC required components. Other vendor-specific extensions are necessary in order to support ANSI RBAC optional components. Fundamental limitations exist, however, due to impracticality of some aspects in the ANSI RBAC standard itself. This paper sets up a framework for assessing implementations of ANSI RBAC for EJB systems.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	Overview of ANSI RBAC . . . . .	2
2.1.1	Reference Model . . . . .	2
2.1.2	Functional Specification . . . . .	4
2.2	Overview of EJB Security . . . . .	4
2.2.1	EJB . . . . .	5
2.2.2	EJB Security Subsystem . . . . .	7
<b>3</b>	<b>Related Work</b>	<b>8</b>
<b>4</b>	<b>EJB Protection State</b>	<b>9</b>
4.1	EJB Access Controls . . . . .	9
4.2	Formalization of the Protection State . . . . .	11
<b>5</b>	<b>Analysis of Support for ANSI RBAC</b>	<b>13</b>
5.1	Reference Model . . . . .	13
5.1.1	Core RBAC . . . . .	13
5.1.2	Hierarchical RBAC . . . . .	14
5.1.3	Constrained RBAC . . . . .	15
5.2	Example . . . . .	15
5.3	Functional Specification . . . . .	18
<b>6</b>	<b>Discussion</b>	<b>21</b>
<b>7</b>	<b>Conclusion</b>	<b>21</b>
	<b>References</b>	<b>22</b>

# 1 Introduction

The American National Standard for Information Technology Role Based Access Control (ANSI RBAC) [ANS04] is a specification of an access control system in which permissions are associated with roles and users are assigned to appropriate roles. RBAC is an approach to address the needs of the commercial enterprises better than lattice-based MAC [BL75] and owner-based DAC [Lam71]. A role can represent competency, authority, responsibility, or specific duty assignments. The ANSI RBAC standard consists of two main parts: the RBAC Reference Model, and the RBAC System and Administrative Functional Specification. Both parts cover four components: the minimum set of features included in all RBAC systems (*Core RBAC*), role hierarchies (*Hierarchical RBAC*), static constraint relations (*Static Separation of Duty Relations*), and dynamic constraint (*Dynamic Separation of Duty Relations*). A major purpose of RBAC is to facilitate access control administration and review.

Many papers propose ways to support or implement RBAC using commercial technologies, e.g., Oracle [Not95], NetWare [ES95], Java [Giu98], DG/UX [Mey97], J2EE [ZSN<sup>+</sup>06, BO06], object-oriented systems [Bar95], object-oriented databases [Won97], MS Windows NT [BC98], enterprise security management systems [Awi97]. Evidence of RBAC becoming a dominant access control paradigm is the approval of the American National Standard Institute (ANSI) RBAC Standard [ANS04] in 2004.

At the same time, commercial middleware technologies—such as Common Object Request Broker Architecture (CORBA) [OMG99], COM+ [Obe00], Enterprise Java Beans (EJB) [DYK01]—became mature, with distributed enterprise applications routinely developed with the use of middleware. Each middleware technology, however, comes with its own security subsystem [Edd99, OMG02, HFB01], sometimes dependent on and specific to the underlying operating system (OS). For instance, COM+ security [Edd99] is tied into Microsoft Windows OS and its services.

The ability of a particular middleware technology to support specific type of access control policy is an open and practical question. It is not a simple question because of the following three reasons.

First, different middleware technologies and their subsystems are defined in different forms and formats. For example, CORBA is specified in the form of open application programming interfaces (APIs), whereas EJB is defined through APIs as well as the syntax and semantics of the accompanying extensible markup language (XML) files used for configuring the EJB container. COM+ is defined through APIs as well as graphical user interfaces (GUI) for configuring the behavior of a COM+ server. The variations in the form, terminology, and format of the middleware definitions lead to the difficulty of identifying the correspondence among the (security and other) capabilities of any two middleware technologies.

Second, the capabilities of the middleware access controls are not defined in the terms of any particular access control model. Instead, the controls are defined in terms of general mechanisms which are supposed to be adequate for the majority of cases and could be configured to support various access control models. Designed to support a variety of policy types as well as large scale and diverse distributed applications, the controls seem to be a result of engineering compromises between, among others, perceived customer requirements, the capabilities of the target runtime environment, and their expected usage. For example, CORBA access controls are defined in the terms of principal's *attributes*, *required*, and *granted rights*, whereas EJB controls are defined using *role mappings* and *role-method permissions*. Assessing the capability of middleware controls to enforce particular types of authorization policies is harder due to the mismatch in the terminology between the published access control models and languages of the controls.

Third, the security subsystem semantics in commercial middleware is defined imprecisely, leaving room for misinterpretation. We clarify the semantics of the security subsystem and analyze its ability to support ANSI RBAC for one particular industrial middleware technology—EJB.

In this paper, we define the protection state of the access control subsystem of EJB. Our definitions offer precise and unambiguous interpretation of the middleware access controls. The

language of the middleware protection state enables the analysis of the access control system on the subject of its support for specific access control models. To demonstrate the utility of the protection state definitions and to aid application developers and owners, we analyzed the degree to which EJB supports the family of role-based access control models as defined by ANSI RBAC Standard [ANS04].

We have formalized the authorization-related parts of EJB v.3.0 [DK06] into protection state configuration through studying its description and specifications. Then, we used the protection state configuration to analyze EJB in regards to its support for ANSI RBAC. When it was possible, we showed how the corresponding ANSI RBAC construct can be expressed in the language of the EJB protection state. In the cases when support for specific ANSI RBAC feature required implementation-dependent functionality, we explicitly stated what needed to be implemented by the middleware developers, or configured by the security administrators. When we could not identify the means of supporting an ANSI RBAC feature, we stated so. We have summarized the results of our analysis at the end of the paper (Section 6).

Our analysis suggests that the EJB specification is not capable of fully supporting even the required Core RBAC component, in order for it to be ANSI RBAC compliant. This is due to the fact that the EJB specification relies on (1) the operational environment to provide the management of user accounts, and the run-time environment to manage (2) user sessions, and (3) role activation. While these limitations can be easily worked around through vendor-specific and implementation dependent extensions, each EJB implementation would have to be evaluated for ANSI RBAC separately. In order to provide standard support for administering and reviewing user accounts, their roles and their sessions, the corresponding administrative interfaces would need to be added to EJB, which would be contrary to the emerging practice of “outsourcing” such functions to enterprise-wide single sign-on and identity management solutions.

This paper establishes a framework for implementing and assessing implementations of ANSI RBAC using EJB. The results provide directions for EJB developers supporting ANSI RBAC in their systems and criteria for users and application developers for selecting those EJB implementations that support required and optional components of ANSI RBAC.

The rest of the paper is organized as follows. Section 2 provides an overview of ANSI RBAC and EJB. Section 3 discusses related work. Section 4 formally defines the protection state of the EJB access control subsystem. Section 5 discusses how an ANSI RBAC based access policy maps to the EJB protection state, and provides an example. We discuss the results of our analysis in Section 6. Section 7 concludes the paper.

## 2 Background

This section provides background on ANSI RBAC and EJB Security that is necessary for understanding the rest of the paper. Readers familiar with both can skip directly to Section 3.

### 2.1 Overview of ANSI RBAC

Role based access control (RBAC) was introduced more than a decade ago [FK92, SCFY96]. Over the years, RBAC has enjoyed significant attention as many research papers were written on topics related to RBAC; and in recent years, vendors of commercial products have started implementing various RBAC features in their solutions.

The National Institute of Standards and Technology (NIST) initiated a process to develop a standard for RBAC to achieve a consistent and uniform definition of RBAC features. An initial draft of a standard for RBAC was proposed in the year 2000 [SFK00]. A second version was later publicly released in 2001 [FSG<sup>+</sup>01]. This second version was then submitted to the International Committee for Information Technology Standards (INCITS), where further changes were made to the proposed standard. Lastly, INCITS approved the standard for submittal to the American National Standards Institute (ANSI). The standard was later approved in 2004 [ANS04]. The ANSI RBAC standard consists of two main parts as described in the following sections.

### 2.1.1 Reference Model

The RBAC Reference Model defines sets of basic RBAC elements, relations, and functions that the standard includes. This model is defined in terms of four major RBAC components as described in the following sections.

#### Core RBAC

Core RBAC defines the minimum set of elements required to achieve RBAC functionality. Core RBAC must be implemented as a minimum in RBAC systems. The other components described below, which are independent of each other, can be implemented separately.

Core RBAC elements are defined as follows [ANS04, pp.4-5]:

#### Definition 1 [Core RBAC]

- $USERS, ROLES, OPS,$  and  $OBS$  (*users, roles, operations, and objects respectively*)
- $UA \subseteq USERS \times ROLES$ , a many-to-many mapping user-to-role assignment relation
- $assigned\_users(r : ROLES) \rightarrow 2^{USERS}$ , the mapping of role  $r$  onto a set of users. Formally:  $assigned\_users(r) = \{u \in USERS \mid (u, r) \in UA\}$
- $PRMS = 2^{(OPS \times OBS)}$ , the set of permissions
- $PA \subseteq PERMS \times ROLES$ , a many-to-many mapping permission-to-role assignment relation.
- $assigned\_permissions(r : ROLES) \rightarrow 2^{PRMS}$ , the mapping of role  $r$  onto a set of permissions. Formally:  $assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$
- $Op(p : PRMS) \rightarrow \{op \subseteq OPS\}$ , the permission to operation mapping, which gives the set of operations associated with permission  $p$
- $Ob(p : PRMS) \rightarrow \{ob \subseteq OBS\}$ , the permission to object mapping, which gives the set of objects associated with permission  $p$
- $SESSIONS =$  the set of sessions
- $session\_users(s : SESSIONS) \rightarrow USERS$ , the mapping of session  $s$  onto the corresponding user
- $session\_roles(s : SESSIONS) \rightarrow 2^{ROLES}$ , the mapping of session  $s$  onto a set of roles. Formally:  $session\_roles(s) \subseteq \{r \in ROLES \mid (session\_users(s), r) \in UA\}$
- $avail\_session\_perms(s : SESSIONS) \rightarrow 2^{PRMS}$ , the permissions available to a user in a session =  $\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$

#### Hierarchical RBAC

This component adds relations to support role hierarchies. Role hierarchy is a partial order relation that defines seniority between roles, whereby a senior role has at least the permissions of all of its junior roles, and a junior role is assigned at least all the users of its senior roles. A senior role is also said to “inherit” the permissions of its junior roles.

The standard defines two types of role hierarchies. These types are shown in Figure 1, and are defined as follows:

- General Role Hierarchies: provide support for arbitrary partial order relations to serve as the role hierarchy. This type allows for multiple inheritance of assigned permissions and users; that is, a role can have any number of ascendants, and any number of descendants
- Limited Role Hierarchies: provide more restricted partial order relations that allow a role to have any number of ascendants, but only limited to one descendant

In the presence of role hierarchy, the following is defined:

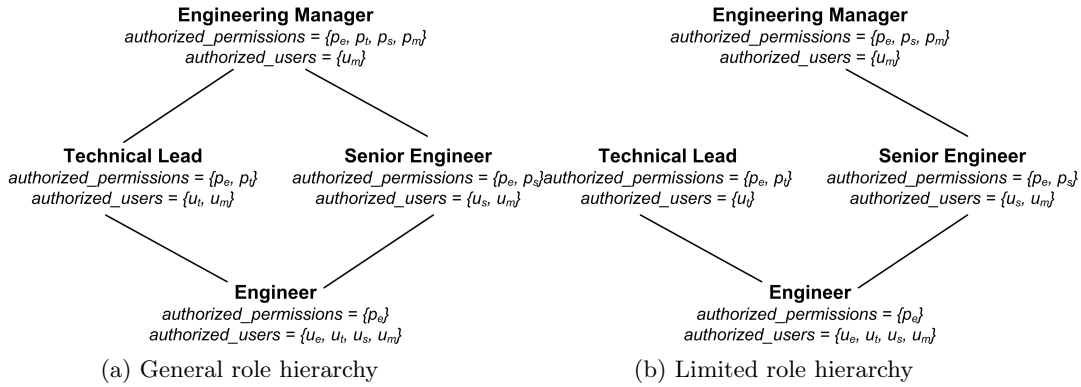


Figure 1: Examples of Hierarchical RBAC

- $authorized\_users(r) = \{u \in USERS | r' \succeq r, (u, r') \in UA\}$  is the mapping of role  $r$  onto a set of users
- $authorized\_permissions(r) = \{p \in PRMS | r \succeq r', (p, r') \in PA\}$  is the mapping of role  $r$  onto a set of permissions

where  $r_{senior} \succeq r_{junior}$  indicates that  $r_{senior}$  inherits all permissions of  $r_{junior}$ , and all users of  $r_{senior}$  are also users of  $r_{junior}$ .

### Constrained RBAC

**Static Separation of Duty (SSD) relations** component defines exclusivity relations among roles with respect to user assignments. **Dynamic Separation of Duty (DSD) Relations** component defines exclusivity relations with respect to roles that are activated as part of a user's session.

#### 2.1.2 Functional Specification

For the four components defined in the RBAC reference model, the RBAC System and Administrative Functional Specification defines the three categories of various operations that are required in an RBAC system. These categories are defined as follows.

The category of *administrative operations* defines operations required for the creation and maintenance of RBAC element sets and relations. Examples of these operations are listed here. A complete list of these operations, as well as their formal definition is included in the standard.

- Core RBAC administrative operations include AddUser, DeleteUser, AddRole, DeleteRole, AssignUser, GrantPermission, and so on
- Hierarchical RBAC administrative operations include AddInheritance, DeleteInheritance, AddAscendant, and AddDescendant
- SSD Relations administrative operations include CreateSsdSet, AddSsdRoleMember, SetSsdSetCardinality, and so forth
- DSD Relations administrative operations include CreateDsdSet, AddDsdRoleMember, SetDsdSetCardinality, and so on

The *administrative reviews* category defines operations required to perform administrative queries on the system. Examples of Core RBAC administrative review functions include RolePermissions, UserPermissions, SessionRoles, and RoleOperationsOnObjects. Other operations for other RBAC components can be found in the standard.

The *system level functionality* category defines operations for creating and managing user sessions and making access control decisions. Examples of such operations are `CreateSession`, `DeleteSession`, `AddActiveRole`, and `CheckAccess`.

## 2.2 Overview of EJB Security

In this section we provide an overview of the EJB architecture, the main parts comprising an EJB system, as well as the declarative and runtime aspects of EJB systems. We also provide a description of the EJB security architecture.

### 2.2.1 EJB

This section provides a brief and informal overview of EJB. More information can be found in the corresponding EJB specification. Readers familiar with EJB are advised to proceed to Section 2.2.2.

The Enterprise Java Beans standard [DK06] defines an architecture for developing and deploying server-side components written in the Java programming language. The EJB architecture specifies the contracts that ensure the interoperability between various EJB components, clients, and deployment environments. These contracts ensure that an EJB product developed by one vendor is compatible with an EJB product provided by another vendor.

The EJB architecture, similar to other middleware technologies, allows application developers to implement their business logic without having to handle transactions, state management, multi-threading, connection pooling, and other deployment platform-dependent issues.

The EJB architecture consists of the following basic parts. These parts are also shown in Figure 2 for `ProductBean`, an example Enterprise Java Bean.

**Enterprise Java Bean** A server-side software component that is composed of one or more Java objects. The enterprise bean exposes certain interfaces that allow clients to communicate with the bean in compliance with the EJB specification. This is shown in Figure 2 as `ProductBean`. The EJB specification [DK06] defines three main types of enterprise beans: *entity*, *session* (which include stateful and stateless session beans), and *message-driven* beans. Depending on the type of the enterprise bean, its functionality ranges from a mere object-oriented abstraction of an entity that exists in persistent storage (such as a record in a database), to a web service implementing certain business logic.

**EJB Container** Provides services—such as persistence, concurrency, bean lifecycle, resource pooling, and security—to the enterprise beans it hosts. Multiple enterprise beans typically exist inside a single container. The container vendor provides necessary tools, which are specific to their container, to help in the deployment of enterprise beans, as well as runtime support for the deployed bean instances.

**EJB Server** Provides runtime environment to one or more containers. Since EJB specification does not explicitly define the separation of roles between containers and servers, they usually come inseparable as one system.

**EJB Client** A software component that invokes methods on the Enterprise Java Bean. The EJB architecture allows a variety of client applications to utilize the business logic that the beans provide. Servlets or Java Server Pages (JSP), Java stand-alone applications or applets are common types of EJB clients. EJBs can also be clients of other EJBs. CORBA-based applications, which are not necessarily developed in Java, may also be clients of EJBs. All EJB clients access enterprise beans logic through predefined protocols and software interfaces. These interfaces define the methods that can be invoked on the bean.

**Remote Business Interface** Java interfaces that are provided by the Enterprise Java Bean and marked with the `@Remote` Java language metadata annotation. [DK06] The EJB container tools handle the generation of required logic in order to support remote access to methods defined by this interface.



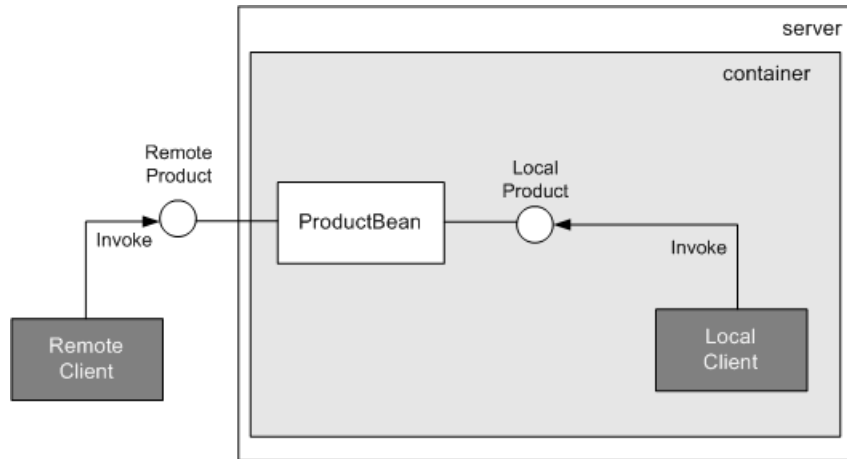


Figure 2: Basic parts of EJB architecture for an example Enterprise Java Bean Product

```
import javax.ejb.Remote;

@Remote public interface Product {
    public float getPrice();
    public void setPrice( float newPrice )
        throws InvalidPriceException;
};
```

Figure 3: Defining a remote interface for the Product enterprise bean (Product.java)

**Local Business Interface** A Java interface that is provided by the Enterprise Java Bean and that supports local access. Clients that utilize this type of interface have to be collocated in the same Java Virtual Machine (JVM) as the Enterprise Java Bean.

Although Enterprise Java Beans are written in the Java programming language, fully compliant EJB deployment environments support the Internet Inter-ORB Protocol (IIOP) [OMG04], leveraging IIOP and the Common Secure Interoperability Protocol Version 2 (CSIV2) [OMG04] capabilities which allow CORBA clients (which can be written in languages other than Java) to access enterprise bean objects.

**Declarative Part** Defining remote, and local interfaces as well as implementing the business logic in EJB is as easy as in standard Java. Figure 3 shows an example of an enterprise bean remote interface definition, and Figure 4 illustrates an example of the corresponding implementation for that interface.

In EJB 3.0, the metadata annotations defined in Java Development Kit (JDK) 5.0 and later are used to create annotated Enterprise Java Beans. The tools provided by the EJB Container vendors utilize these annotations to automatically generate proper Java classes as well as other required EJB interfaces.

As an alternative to metadata annotations, a bean developer can also specify transactional, security, and other requirements for the application using the *deployment descriptor*—an XML file with predefined syntax that holds all the explicit metadata for the assembly. The descriptor can be later augmented and altered by an application *assembler* and *deployer*, who play specific roles in the life cycle of enterprise beans predefined by the EJB specification.

```
import javax.ejb.Stateful;

@Stateful public class ProductBean implements Product {
    private float price = 0;

    public float getPrice() {
        return price;
    }

    public void setPrice( float newPrice ) {
        if ( price < 0 ) {
            throw new InvalidPriceException();
        }

        price = newPrice;
    }
}
```

Figure 4: Implementing the remote interface for the Product enterprise bean (Product-Bean.java)

**Runtime Part** While the remote object model for EJB components is based on the Remote Method Invocation (RMI) API [Sun07], all invocations between J2EE components are performed using IIOP. The use of the RMI remote invocation model over the IIOP protocol is usually referred to as RMI-IIOP. When EJB components use the RMI-IIOP (mandatory for EJB 2.0 and higher), the standard mapping of the EJB architecture to CORBA enables interoperability with multi-vendor ORBs, other EJB servers, and CORBA clients written in languages other than Java.

Because of the IIOP, the same object reference used for CORBA is used in the EJB. The similarities between CORBA and EJB lie in their use of a secure channel, as well as their client and server security layer architectures. For more detailed explanation of EJB technology please refer to [RSB05].

### 2.2.2 EJB Security Subsystem

The EJB protection architecture is conceptually simple: When the client program invokes a method on a target EJB object, the identity of the subject associated with the calling client is transmitted to the EJB object's container. The container checks whether the calling subject has a right to invoke the requested method. If so, the container permits the invocation on the method.

#### Client Security Service

Because of the use of IIOP and CSiv2, the responsibilities of an EJB client security service (CSS) are similar to those of a CORBA CSS:

1. creating a secure channel with the target security service (TSS), and
2. obtaining the user's authenticated credentials or passing username and password over the CSiv2 context to TSS, as well as
3. protecting request messages and verifying response messages.

Treated by the EJB specification as an integral part of the server container, a TSS establishes and maintains a secure channel with the clients, verifies authenticated credentials or performs client authentication itself, implements message protection policies, and performs access checks

before an invocation is dispatched to an enterprise bean. Depending on the application configuration, which is done through the deployment descriptor, the container associates the runtime security context of the dispatched method either with the identity of the calling client or with some other subject. Other security-related responsibilities of a container include the following:

- Isolating the enterprise bean instances from each other and from other application components running on the server,
- Preventing enterprise bean instances from gaining unauthorized access to the system information of the server and its resources,
- Ensuring the security of the persistent state of the enterprise beans,
- Managing the mapping of principals on calls to other enterprise beans, or on access to resource managers, according to the defined security policy,
- Allowing the same enterprise bean to be deployed independently multiple times, each time with a different security policy.

### Implementation of Security Functions

The security parts of EJB specification focus largely on authentication and access control. The specification relies on CSIv2 level 0 for message protection, and it leaves support for security auditing to the discretion of container vendors. We describe EJB access control architecture in Section 4.1.

**Authentication** User authentication is either performed by the client's infrastructure (such as Kerberos), or by the EJB server itself. In the latter case, the EJB server receives user authentication data (only username and password for CSIv2 level 0) or credentials from a client and authenticates the client using local authentication service, which is not predefined by the specification. Once the container authenticated the client (or verified its credentials), it enforces access control policies. The notion of a *principal* is used in the EJB specification to refer to authenticated clients.

**Administration** Some security administration tasks of EJB servers are performed through changes in deployment descriptors. This includes definition of security roles, method permissions, and specification of security identity, either delegated or predetermined, for dispatching calls to bean methods. Other tasks, such as mapping users to roles, specifying message protection, administering an audit, and authentication mechanisms, are beyond the scope of the EJB specification and are therefore left up to the vendors of container products and deployment tools.

## 3 Related Work

Over the past decade, there has been no shortage of papers describing ways to model or implement RBAC using the technologies employed by the commercial users.

RBAC implementations employing constructs built into operating systems include [SA98], [AS01], and [Fad99]. [SA98] describes an implementation of RBAC for UNIX based systems; [AS01] illustrates a similar method for implementing RBAC for Windows based systems. In both implementations, operating system users and permissions are used as is, but groups are used to represent RBAC roles. In addition, an extra administrative software layer is used to maintain role hierarchy. [Fad99] describes a different implementation of RBAC for UNIX based systems, which implements RBAC roles using UNIX user accounts.

In the area of database management systems, [RS98] analyzes and compares RBAC features implemented in IBM Informix Dynamic Server 7.2, Sybase Adaptive Server 11.5, and Oracle Enterprise Server 8.0.

RBAC has also been implemented in Web based and distributed systems. An implementation of RBAC for NFS [IET03] is described in [GDS97]. In this implementation the remote file system is mounted as usual, but then access checks to remote files are overridden with RBAC decisions.

Other implementations, such as the ones described in [Bar97] and [Gut01], use the Lightweight Directory Access Protocol (LDAP) [WHK97] to implement RBAC. There are two main differences between the two implementations in [Bar97] and [Gut01]. The first difference is related to authentication: the implementation in [Gut01] makes no restrictions on how authentication is accomplished, while [Bar97] uses X.509 [IT00] certificates for authentication. The second main difference is in the LDAP schema used to represent users, roles, and permissions. In [Bar97], a user object points to a role, which in turn points to a permission, while in [Gut01], a user object points to a role, where permissions are saved.

Other designs, such as those presented in [CO02] and [ZM04], use the Attribute Certificates (ACs) defined in the Privilege Management Infrastructure (PMI) [IT00] to implement RBAC. [ZM04] use PKI for authentication, while [CO02] makes no such restriction. In addition, the former is designed to support multiple policies.

Web based implementations include [Giu99], [FBK99], and [PSA01]. [Giu99] describes an architecture that provides RBAC on the World Wide Web using server-side Java technologies. This architecture describes how RBAC can be implemented using the Java Authentication and Authorization Service (JAAS) [Sun01]. [FBK99] provides another design that employs Common Gateway Interface (CGI) to implement RBAC. [PSA01], on the other hand, presents two architectures for implementing RBAC on the Web: user-pull and server-pull. These two architectures are implemented using either cookies, X.509, Secure Socket Layer (SSL) [FKK96], or LDAP.

Finally, middleware based RBAC implementations include [Ahn00] and [OF02]. [Ahn00] describes a method for integrating RBAC into DCOM. In this implementation, roles are mapped to Windows NT user groups. The paper proposes an RBAC monitor, which is responsible for enforcing RBAC policies. On the other hand, [OF02] present an implementation of RBAC for CORBASec. The implementation uses PoliCap [WdSFW<sup>+</sup>02], which was developed to be part of JaCoWeb [WF99]; the latter is a project for defining and implementing authorization schemes for distributed systems based on CORBASec and Java.

## 4 EJB Protection State

In this section, we first introduce the EJB access control architecture. Then, we formally define a configuration of the EJB protection state.

### 4.1 EJB Access Controls

An EJB container controls access to its beans at the level of an individual method on a bean class, although not a bean instance. That is, if different instances of the same bean have different access control requirements, they should be placed in different application assemblies, which are defined by JAR files. This means that the scope of the EJB's policy domain is the application assembly.

The EJB access control architecture provides two ways for enforcing access control decisions. One approach, known in EJB terminology as *declarative security*, is to configure the container to enforce authorization policy. The other is achieved by coding authorization decision and enforcement logic into the bean methods. In the former case, access permissions of principals are defined either using deployment descriptors, or through code annotations. The declarative approach decouples business logic from security logic. In the latter approach, known as *programmatically security*, the application developers employ method calls `isCallerInRole` and `getCallerPrincipal` to obtain the information about the caller in order to enforce those access control policies that cannot be expressed using the declarative approach.

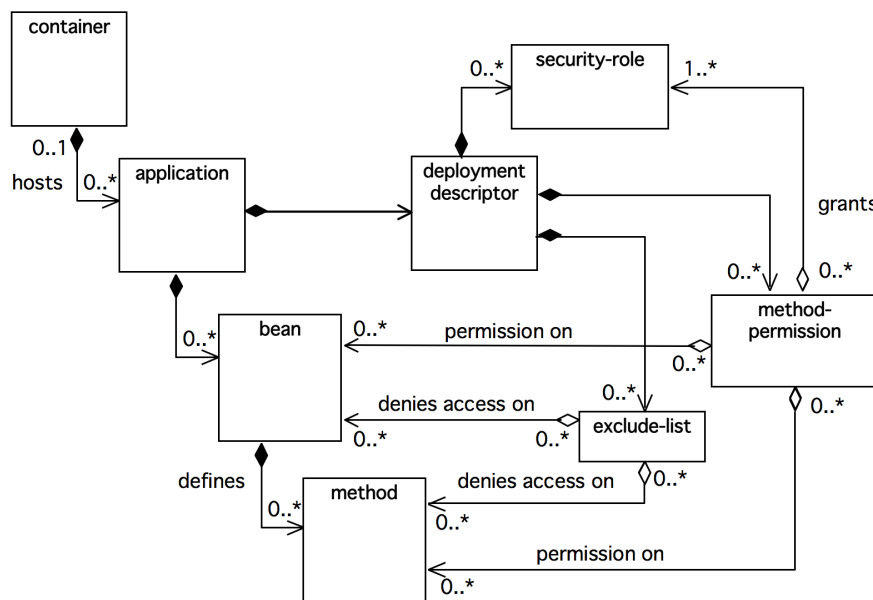


Figure 5: Relationships among the sections of deployment descriptor used for expressing access control policy and the elements of an EJB application

Authorization to invoke enterprise bean’s methods is enforced by the container. It grants or denies clients’ requests to execute the methods in conformance with access control policies described in the *deployment descriptor* and/or through bean’s metadata annotations. Since bean’s metadata annotations are equivalent in the expressiveness to the policies supported by the deployment descriptor, we use only the latter in the rest of the paper.

Access control decisions are based on the *security roles* (or just “roles” for short) of the principal that represents the calling client. The security role is defined in the EJB specification as “a semantic grouping of permissions that a given type of users of the application must have in order to successfully use the application” [DK06, page 456]. As defined by the specification, there are three types of deployment descriptor sections relevant to the declarative access control: **security-role**, **method-permission**, and **exclude-list**. The **exclude-list** section lists those methods that cannot be called by any principal, no matter which roles the principal has. Figure 4.1 uses Unified Modeling Language (UML) [OMG07a, OMG07b] notation to summarize the relationships among authorization-related sections of the deployment descriptor and the elements of an EJB application. In the rest of this section, we describe syntax and semantics of the two other sections.

Each **security-role** section lists a role with optional human-readable unstructured description of the role. This role can be referenced in other sections of the deployment descriptor. In essence, these sections define a set of roles for an EJB application.

Assignment of permissions to roles is done in **method-permission** sections. Each such section lists roles permitted to invoke one or more methods. Special role name “unchecked” can be used to indicate that all roles are permitted to invoke the listed method(s). Each method is defined by the name of the bean class, method name, and, optionally, the formal parameter types to distinguish methods with overloaded names. Special method name “\*” refers to all methods on

Roles	Methods
$r_1$	$b_1 \cdot m_1$
$r_1, r_2$	$b_1 \cdot m_2, b_1 \cdot m_4$
“unchecked”	$b_2 \cdot m_1, b_3 \cdot *$

Table 1: Examples of method-permission sections of EJB deployment descriptor. For clarity sake, the data representation is converted from XML notation to human-understandable form, with each row corresponding to an individual section.

a given bean.

An example of an assignment done through `method-permission` sections is shown in Table 4.1. First row illustrates an assignment of a permission to invoke method  $m_1$  on bean  $b_1$  ( $b_1 \cdot m_1$ ) to role  $r_1$ . The second row shows how several roles ( $r_1$  and  $r_2$ ) can be granted permissions to invoke any of the listed methods ( $b_1 \cdot m_2$  and  $b_1 \cdot m_4$ ). This means that any principal that has any of these two roles can invoke any of these two methods. Last row provides an example of using “unchecked” and “\*” keywords. It states that any principal can invoke method  $b_2 \cdot m_1$  as well as any method on bean  $b_3$ . The overall set of methods a principal can invoke on a given EJB application is the union of all the methods the principal’s roles are permitted to invoke. For example, if a deployment descriptor contains only the three method-permission sections listed in Table 4.1, then a principal with role  $r_2$  is granted permission to invoke methods  $b_1 \cdot m_2$ ,  $b_1 \cdot m_4$ ,  $b_2 \cdot m_1$ , and any method on bean  $b_3$ .

If a method (1) is not listed in any of the `method-permission` and `exclude-list` sections of a deployment descriptor, and (2) has no `@DenyAll` annotation in the code, then it is accessible by any principle.<sup>1</sup> For instance, if  $b_1 \cdot m_3$  is such a method then any principle would be able to invoke it.

Even though the syntax of `method-permission` section allows listing more than one role and method, we will assume without the loss of generality that each section contains only one role and one method, as shown in the first row of Table 4.1. It is easy to define an algorithm for converting any number of `method-permission` sections in a deployment descriptor to this form. This assumption will simplify the definition of the protection state and the algorithm for making access control decisions in Section 4.2.

In addition to the above deployment descriptor sections, EJB server vendors (or container providers) define container-specific sections of deployment descriptors that map users and/or groups to roles. Table 2 shows additional deployment descriptor sections for major commercial EJB servers. Since the notions of users, groups, and the mapping from them to roles are lacking from the EJB v3.0 specification, these vendor-specific additions to the EJB system will not be used for defining the EJB protection state.

## 4.2 Formalization of the Protection State

In this section, we formalize the semantics of the EJB access control architecture.

**Definition 2 [EJB Protection State]** *A configuration of an EJB system protection state is a tuple  $(R, B, M, MP, X)$  interpreted as follows:*

- $R$  is a set of roles defined in the `assembly-descriptor` part of of the deployment descriptor provided with the EJB application. These roles are defined using the `security-role` tags. This set also includes special role “unchecked”.
- $B$  is a set of enterprise beans listed in the `enterprise-beans` section of the deployment descriptor.

<sup>1</sup>According to Section 17.3.2.3 of the EJB specification [DK06], methods with unspecified permissions must be treated by the container as “unchecked.”

App. Server	Section(s)	Comments
Oracle	users, groups	A <i>security-role-mapping</i> XML tag maps logical roles defined in the application deployment descriptor to entities defined in the <i>users</i> and <i>groups</i> sections
Sun ONE	principal-name, group-name	A <i>security-role-mapping</i> tag defines mapping between <i>principal-names</i> and roles, and/or between <i>group-names</i> and roles
BEA WebLogic	principal-name	A <i>security-role-assignment</i> XML tag declares mapping between <i>principal-names</i> and roles
IBM WebSphere	users, groups	Tools establish user-group memberships and mapping between groups and roles

Table 2: Additional authorization-related sections used in deployment descriptors of commercial EJB servers

- *OPS* is a set of methods defined by the enterprise beans of the application. Members of this set are denoted as  $m_i$ . The set also includes special method “\*” for any bean defined by the application and signifying any method on that bean; for example,  $OPS = \{m_1, m_2, \dots\} \cup \{*\}$ .
- $M \subseteq B \times OPS$  is the set of available uniquely identifiable methods. Members of this set are denoted  $b_i \cdot m_j$ .
- $MP \subseteq R \times M$  is a many-to-many permission assignment of EJB application roles to invoke methods, as specified in `method-permission` sections of the application’s deployment descriptor.
- $X \subseteq M$  is a subset of methods—defined by `exclude-list` sections of the deployment descriptor—invocation of which is denied to any role.

Note that the implementations of EJB containers and servers commonly have extensions to the deployment descriptors, which enable defining sets of users and groups, as well as assigning them to roles. Such vendor-specific extensions result in additional elements of the protection state. However, all elements defined in Definition 2 are present in any EJB implementation compliant with the specification. When analyzing in Section 5 EJB support for RBAC, we will identify additional elements of EJB protection state that are necessary for the support.

Given the protection state of an EJB application, Algorithm 1 defines the outcome of an access control decision. First, a check is performed on the membership of the requested method in the list of blocked methods. If the method is found in the list, then the access is denied. If not, then the method permissions are checked for every role of the principal and the special role “unchecked.” If no appropriate element is in  $MP$ , then access is denied.

## 5 Analysis of Support for ANSI RBAC

As described in Section 2.1, the ANSI RBAC Reference Model defines four major components. In order for a system to conform to ANSI RBAC, Core RBAC must be implemented at a minimum. ANSI compliant RBAC system can also implement Hierarchical RBAC, which defines hierarchies

```

Authorize( $p : 2^R, b_i \cdot m_j : M$ )  $\rightarrow$  {allow, deny}
if  $b_i \cdot m_j \in X$  then
  return deny
end if
for all  $r \in p \cup \{\text{"unchecked"}\}$  do
  if  $(r, b_i \cdot m_j) \in MP \vee (r, b_i^*) \in MP$  then
    return allow
  end if
end for
return deny

```

**Algorithm 1:** Authorization decision in EJB. Decide authorization for principal  $p \equiv \{r_1, r_2, \dots, r_n\}$  invoking method  $m_j$  on bean  $b_i$ , where  $r_1, r_2, \dots, r_n \in R$ , and  $b_i m_j \in M$

of roles in addition to everything Core RBAC has. The other two optional components of the standard, Static Separation of Duty (SSD) and Dynamic Separation of Duty (DDS), define relations among roles with respect to user assignments as well as role activation in user sessions.

In Section 5.1, we first examine the extent to which an EJB protection state—as formalized in Definition 2—can support each of the four ANSI RBAC model components. In Section 5.2 we provide an example that illustrates the abilities of an EJB system to support ANSI RBAC. In Section 5.3 we then analyze the degree to which the structures defined in EJB specification support the functional specification of ANSI RBAC. In Section 6 we discuss results of our analysis.

## 5.1 Reference Model

### 5.1.1 Core RBAC

Various Core RBAC data elements are mapped readily into EJB using the sets defined in Section 4. For example, the *ROLES* set in RBAC maps directly to  $R$ , which defines the EJB security roles; RBAC objects (*OBJ*) are equivalent to EJB beans ( $B$ ); RBAC operations (*OPS*) are represented by EJB *OPS*. The representation of other relations defined in Core RBAC is outside the scope of the EJB standard, as we will discuss later in this section. We first define Core RBAC in the language of the EJB protection system more formally as follows:

**Definition 3 [Core RBAC in EJB]** *Core RBAC in the language of EJB is defined by the EJB system protection state outlined in Definition 2, as well as the following additional elements:*

- *USERS* is the set of users, where members of this set are defined in the operational environment of the EJB system.
- *ROLES* =  $R$ , is the set of roles as defined in Definition 2.
- *OBS* =  $B$  is a set of enterprise beans.
- $UA = USERS \times ROLES$ , is a many-to-many assignment of users to roles.
- $assigned\_users(r : ROLES) = \{u \in USERS \mid (u, r) \in UA\}$ , is a function that returns the set of users in *USERS* that are assigned to the given role  $r$ .
- $PRMS \subseteq M - X$ , is a set of permissions to invoke EJB methods provided that these methods do not exist in the exclusion set  $X$ . The existence of  $b_i \cdot m_j$ , or  $b_i \cdot *$  in *PRMS* provides permission to invoke a specific method  $m_j$ , or all methods on bean  $b_i$ , respectively.
- $PA \subseteq PRMS \times ROLES$ , a many-to-many assignment of permissions to roles.
- $assigned\_permissions(r : ROLES) = \{p \in PRMS \mid (p, r) \in PA\}$ , is a function that returns the set of permissions in *PRMS* that are assigned to the given role  $r$ .



- $Op(p : PERMS) \rightarrow \{op \in OPS\}$ , a function that returns a set of operations that are associated with the given permission  $p$ .
- $Ob(p : PERMS) \rightarrow \{ob \in OBS\}$ , a function that returns a set of objects that are associated with the given permission  $p$ .
- $SESSIONS$  is a set of sessions for a specific application. Members of this set are mappings between authenticated users and their activated roles for a specific EJB application.
- $session\_users(s : SESSIONS) \rightarrow USERS$ , the mapping of session  $s$  onto the corresponding user.
- $session\_roles(s : SESSIONS) \rightarrow 2^{ROLES}$ , the mapping of session  $s$  onto a set of roles. Formally:  $session\_roles(s_i) \subseteq \{r \in ROLES \mid (session\_users(s_i), r) \in UA\}$ .
- $avail\_session\_perms(s : SESSIONS) \rightarrow 2^{PRMS}$ , the permissions available to a user in a session =  $\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$ .

In order to support Core RBAC in EJB systems, Definition 3 identifies additional elements to those identified in Definition 2. These additional elements are related to users and sessions. In the rest of this section we discuss how elements of Definition 3 are or can be supported in an EJB system.

Although the EJB standard [DK06] does not mandate how users must be supported in an EJB system, various implementations of EJB servers and containers implement extensions to deployment descriptors, as described to in Section 4.2. These extensions provide support for adding users to the system, as well as mapping those users to roles. The  $USERS$  set in Definition 3 abstracts this support; however, this support is implementation-dependent. By the same token, support for  $UA$  and  $assigned\_users$  is also implementation-dependent.

The  $SESSIONS$  set is another element of Definition 3. In relation to support for users, the EJB standard does not specify a mapping of authenticated users to roles, or more precisely, role activation. Hence, EJB server's support for sessions is outside the scope of the EJB standard and is implementation-dependent. Similarly, in order to fully support Core RBAC, EJB implementations' support for session-related functions such as  $session\_users$ ,  $session\_roles$ , and  $avail\_session\_perms$  are outside the scope of the EJB standard.

On the other hand, the sets  $ROLES$ ,  $OPS$ , and  $OBS$ ; the relations  $PRMS$  and  $PA$ ; and the functions  $Op$  and  $Ob$  are all supported by the EJB standard as these can be readily obtained from the deployment-descriptor.

To summarize, about half the elements of ANSI Core RBAC can be provided by any implementation compliant with the EJB standard; however, support for  $USERS$ ,  $UA$ ,  $assigned\_users$ ,  $SESSIONS$ ,  $session\_users$ ,  $session\_roles$ , and  $avail\_session\_perms$ , which relate to users and sessions, if provided, can only be implementation-dependent.

### 5.1.2 Hierarchical RBAC

The Hierarchical RBAC component specifies two types of role hierarchies: *general* and *limited*. Both types are formally defined using elements of Core RBAC. In addition to role hierarchies, Hierarchical RBAC defines two functions:  $authorized\_users$  and  $authorized\_permissions$ , as described in Section 5.1.2. Although the EJB standard does not provide direct support for Hierarchical RBAC, an EJB implementation can still emulate both types of role hierarchies. The rest of this section discusses ways of emulating Hierarchical RBAC in EJB.

EJB server administrative tools can be modified in order to support role hierarchy. First, the administrative tools must maintain hierarchy relationships between roles in a repository. Second, the tools must ensure that when method permissions are granted to a certain role in a deployment descriptor, those method permissions are also appropriately and consistently granted to all junior roles. Finally, the administrative tools must also keep track of whether a permission has been directly assigned to a role or the role inherited this permission through a role hierarchy. No special run-time support for role hierarchies would then be needed. This

approach is similar to the ones used in [AS01] and [SA98] in order to support role hierarchy in various operating systems.

An alternative is an approach in which inherited permissions are determined at run-time. This approach would require the EJB server—or more specifically the Target Security Service (TSS) described in Section 2.2.2—to examine the role hierarchy repository during run-time. A certain role is then granted permission to invoke a specific method not only based on direct permission-to-role assignment, but also based on permissions granted to a junior role. In addition to a repository that maintains role hierarchy relationship, a run-time computation of inherited permissions would be required. A similar approach is adopted in [FBK99] for Common Gateway Interface (CGI) based Web applications, and in [Giu99] for Java Authentication and Authorization Service (JAAS) [Sun01] based access control.

With either of the above approaches, support for this role hierarchy—and the *authorized\_users* and *authorized\_permissions* functions required for Hierarchical RBAC—is implementation-dependent and is not specified by the EJB standard.

### 5.1.3 Constrained RBAC

The Constrained RBAC component introduces separation of duty relations to the RBAC reference model. As with Hierarchical RBAC, these relations are defined in terms of Core RBAC constructs. In essence, SSD constrains user-to-role assignment (*UA* set and *assigned\_users* function) and the role hierarchy (*RH* set and *authorized\_users* function). DSD, on the other hand, constrains the role activation (*SESSIONS* set and *session\_roles* function). Since user accounts, role hierarchies, and role activation are beyond the scope of EJB, the Constrained RBAC component, if supported, would have to be implementation-dependent.

## 5.2 Example

In this section we present an example that illustrates the abilities of an EJB system to support ANSI RBAC. As discussed in Section 5.1.2, the EJB standard does not provide direct support for role hierarchy; however, emulation of such support is possible as discussed earlier, and is straightforward. Hence, role hierarchy is not illustrated in this example.

The example in this section consists of a simple system that maintains employee and engineering project records in an engineering company. The system allows different users to perform various operations on the project and employee records, based on the users' roles in the company. The system handles the manipulation of various records through enterprise beans of two types: `EngineeringProject` and `Employee`. These enterprise beans are depicted in Figure 6. The figure shows the methods that can be invoked on the beans. The system also defines seven different user roles. Based on these roles and according to the policies listed in Figure 7, users are allowed to invoke various methods on a specific EJB. These roles are defined as follows:

- *Employee* represents a company employee.
- *Engineering Department* represents an employee of the engineering department.
- *Engineer* performs various engineering tasks in the company.
- *Product Engineer* is responsible for managing a product line.
- *Quality Engineer* is a quality assurance engineer.
- *Project Lead* oversees and leads the development of a project.
- *Director* is an engineering department director.

**Action Item 1** *Kosta: Wesam to make terminology consistent in this section: use either “product engineer” or “production engineer.”*

**Action Item 2** *Kosta: Wesam to consider if the methods for both example beans should follow Java naming convention, `makeChanges`, instead of `make_changes`. If so, the method names everywhere in the example have to adjusted accordingly.*

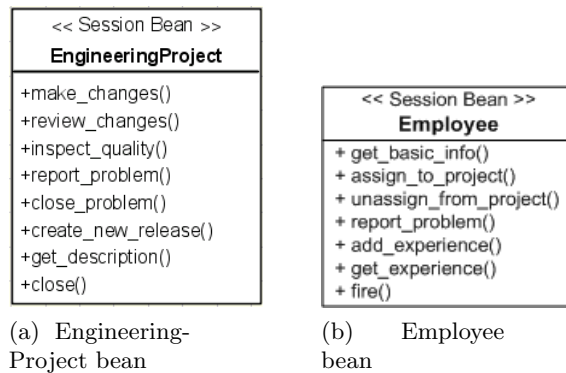


Figure 6: Example session beans

1. Anyone in the organization can look up an employee's basic information, such as their name, department, phone number, and office location.
2. Everyone in the engineering department can get a description of and report problems regarding any project and look up experience of any employee.
3. Engineers, assigned to projects, can make changes and review changes related to their projects.
4. Quality engineers, in addition to being granted engineers' rights, can inspect the quality of projects they are assigned to.
5. Product engineers, in addition to possessing engineers' rights, can create new releases.
6. The project lead, in addition to possessing the rights granted to production and quality engineers, can also close problems.
7. The director, in addition to being granted the rights of project leads, can manage employees (assign them to projects, un-assign them from projects, look up experience, add new records to their experience, and fire them) and close projects.

Figure 7: Authorization policy for the example EJB system describing what actions are allowed. All other actions are denied.

Roles	Methods													
	EngineeringProject Bean								Employee Bean					
	make_changes()	review_changes()	inspect_quality()	report_problem()	close_problem()	create_new_release()	get_description()	close()	get_basic_info()	assign_to_project()	unassign_from_project()	add_experience()	get_experience()	fire()
Employee									✓				✓	
Engineering Department				✓			✓		✓				✓	
Engineer	✓	✓							✓				✓	
Product Engineer						✓			✓				✓	
Quality Engineer			✓						✓				✓	
Project Lead					✓				✓				✓	
Director							✓	✓	✓	✓	✓	✓	✓	✓

Table 3: Permission-to-role assignment for the example

User	Group
Alice	accounting
Bob	hardware
Carol	software
Dave	software
Eve	software
Fred	management

Table 4: Example users, groups, and group memberships

The access control policy that defines what actions each role is allowed to perform are summarized in Table 3, where a check mark (“✓”) denotes a granted permission for a specific EJB role to execute the corresponding enterprise bean method. Table 4 shows an example of system users, and their group memberships. Tables in Figure 8 show examples of user-to-role and group-to-role assignments. The following is a formalization of this example system’s protection state as in Definition 2.

- $R = \{\text{Employee, Engineering Department, Engineer, Product Engineer, Quality Engineer, Project Lead, Director}\}$
- $B = \{\text{EngineeringProject, Employee}\}$
- $OPS = \{\text{make\_changes, review\_changes, inspect\_quality, report\_problem, close\_problem, create\_new\_release, get\_description, close, get\_basic\_info, assign\_to\_project, unassign\_from\_project, add\_experience, get\_experience, fire}\}$
- $M = \{\text{EngineeringProject.make\_changes, EngineeringProject.review\_changes, EngineeringProject.inspect\_quality, EngineeringProject.report\_problem, EngineeringProject.close\_problem, EngineeringProject.create\_new\_release, EngineeringProject.get\_description, EngineeringProject.close, Employee.get\_basic\_info, Employee.assign\_to\_project, Employee.unassign\_from\_project, Employee.add\_experience, Employee.get\_experience, Employee.fire}\}$

User	Role
Alice	Employee
Bob	Engineer
Carol	Quality Engineer
Dave	Product Engineer
Eve	Project Lead
Fred	Director

(a) User-to-role assignment

Group	Role
hardware	Engineering Department
software	Engineering Department

(b) Group-to-role mapping

Figure 8: Example EJB system role mappings

- $MP = \{$  (Employee, Employee.get\_basic\_info),  
 (Employee, Employee.get\_experience),  
 (Engineering Department, EngineeringProject.report\_problem),  
 (Engineering Department, EngineeringProject.get\_description),  
 (Engineering Department, Employee.get\_basic\_info),  
 (Engineering Department, Employee.get\_experience),  
 (Engineer, EngineeringProject.make\_changes),  
 (Engineer, EngineeringProject.review\_changes),  
 (Engineer, Employee.get\_basic\_info),  
 (Engineer, Employee.get\_experience),  
 (Product Engineer, EngineeringProject.create\_new\_release),  
 (Engineering Department, Employee.get\_basic\_info),  
 (Engineering Department, Employee.get\_experience),  
 (Quality Engineer, EngineeringProject.inspect\_quality),  
 (Quality Engineer, Employee.get\_basic\_info),  
 (Quality Engineer, Employee.get\_experience),  
 (Project Lead, EngineeringProject.close\_problem),  
 (Project Lead, Employee::get\_basic\_info),  
 (Project Lead, Employee::get\_experience),  
 (Director, EngineeringProject.close),  
 (Director, Employee.get\_basic\_info),  
 (Director, Employee.assign\_to\_project),  
 (Director, Employee.unassign\_from\_project),  
 (Director, Employee.add\_experience),  
 (Director, Employee.get\_experience),  
 (Director, Employee.fire)  $\}$
- $X = \Phi$

The  $R$  and  $B$ sets contain the roles and beans defined in the system.  $OPS$  defines all operations available to various roles. These methods are further qualified by the  $M$  set, where each method is qualified with the name of the bean for which it is defined. The  $MP$  set represents Table 3, and as described in Section 4.2,  $MP$  is a many-to-many permission assignment of EJB application roles to invoke defined methods. These permissions are listed in the `method-permission` sections of the application's deployment descriptor. This example does not require any methods to be in the `exclude-list` sections of the deployment descriptor for the application; hence, set  $X$  is empty.

We use the above formalization of the example system's protection state in order to support the ANSI Core RBAC reference model. Considering Definition 3, the content of  $ROLES$ ,  $OPS$ , and  $OBS$  is straightforward. The rest of the sets are defined as follows.

- $USERS = \{$  Alice, Bob, Carol, Dave, Eve, Fred, accounting, hardware, software, manage-

ment}

- $UA = \{ (Alice, Employee), (Bob, Engineer), (Carol, Quality Engineer), (Dave, Product Engineer), (Eve, Project Lead), (Fred, Director), (hardware, Engineering Department), (software, Engineering Department), (Bob, Engineering Department), (Carol, Engineering Department), (Dave, Engineering Department), (Eve, Engineering Department) \}$
- $PRMS = M$
- $PA = MP$

The EJB 3.0 standard does not specify how EJB roles should be mapped to the user groups and accounts that exist in the bean's operational environment. This makes the *USERS* and *UA* sets dependent solely on the EJB container's operational environment, and the way users are managed there. For example, the *UA* set contains assignments that exist only due to user-group memberships. In this example, Carol is assigned to the *Engineering Department* role through her software group membership.

### 5.3 Functional Specification

This section reports on the results of our analysis of the support that the EJB standard [DK06] can provide for ANSI RBAC system and administrative functional specifications. For the purpose of this analysis, we examined every function specified in Section 6 of [ANS04] on the subject of its support by an EJB container conforming to the EJB standard.

Results of our examination suggest that the software interfaces that the EJB standard mandates are insufficient for implementing most of ANSI RBAC functions as is. Furthermore, the XML data structures defined in the EJB deployment descriptor, are incapable of fully supporting an ANSI RBAC compliant system. These data structures can provide support for implementing a limited number of Core RBAC functions. Other system and administrative Core RBAC functions, as well as all additional functions for Hierarchical and Constrained RBAC, cannot be supported without extending an EJB system implementation beyond what the EJB standard defines.

The following is an examination of various Core RBAC functions and their level of support in the EJB standard.

**AddUser, DeleteUser** operations allow users to be added to the *USERS* set and to be removed from it. In an EJB environment, these are realized in a implementation-dependent manner. For example, the IBM WebSphere Application Server [SCH<sup>+</sup>04] allows EJB application deployers to use various user registries to maintain the *USERS* set. WebSphere can be configured to use the local operating system user accounts, an LDAP [WHK97] server, or a custom user registry.

**AddRole, DeleteRole** add roles to and delete roles from the RBAC system. EJB data structures provide direct support for implementing these functions. They can be implemented by adding or removing a role definition using the `security-role` tags in the `assembly-descriptor` section of the deployment descriptor file.

**AssignUser, DeassignUser** allow assignment relationships to be established between roles and users. Similar to the *AddUser* and *DeleteUser*, these operations need to be implemented in an implementation-dependent manner.

**GrantPermission, RevokePermission** allow invocation permissions to be granted to or revoked for a certain role. These operations can be implemented by adding or removing the corresponding `method-permission` section of the deployment descriptor.

**CreateSession, DeleteSession, AddActiveRole, DropActiveRole** allow for the creation and deletion of sessions, as well as activation of user roles. In an EJB environment, these operations are likely to be implemented in a proprietary manner and would differ from one EJB application server to another.

**CheckAccess** make an access control decision. The `Authorize` method in Algorithm 1 can be used to implement `CheckAccess`.

**AssignedUsers, AssignedRoles** return users assigned to a given role, and roles assigned to a given user, respectively. Since these functions are not supported in EJB 3.0, they need to be provided by the EJB application server.

### Advanced Review Functions for Core RBAC

**RolePermissions** returns the permissions granted to a given role. This function can be implemented by examining the `method-permission` sections, where method permissions are granted to roles.

**UserPermissions** returns permissions assigned to users. Given the permissions assigned to roles (using the *RolePermissions* function), and knowing the roles the user is assigned to (using *AssignedUsers*), the implementation of this function is straightforward.

**SessionRoles, SessionPermissions** return the roles and permissions associated with a specific user session. These can be provided by the EJB application server assuming that the server implementation already support the notion of sessions.

**RoleOperationsOnObject, UserOperationsOnObject** return a set of operations that can be invoked on an object given a certain role or a certain user, respectively. The operations that a certain role is permitted to invoke can be obtained directly from the `method-permission` sections of the deployment descriptor. The operations that a user is permitted to invoke, on the other hand, can be obtained given the implementation of the *RoleOperationsOnObject* as well as the *AssignedRoles* functions.

Table 5 provides a summary of the above results. The table classifies support for ANSI Core RBAC functions in two main categories. The first category contains functions that are supported directly by EJB data structures, whereas the second category identifies the supplemental components that must be implemented in an EJB system—outside the scope of the EJB specifications—in order to support the specified ANSI Core RBAC functions. These components are identified as related to user management, session and role activation. The user management related components are required to handle the addition/deletion of users from the system, as well as user-to-role assignments. On the other hand, the session and role activation related components are required to handle the management of user sessions and activation of permissions.

## 6 Discussion

The results of our analysis suggest that the EJB functionality—as defined through the data structures and interfaces—falls short of fully supporting ANSI RBAC without resorting to vendor-specific extensions. Even in the case of Core RBAC alone—the mandatory part of any compliant implementation of ANSI RBAC—there are two major causes of this inadequacy.

The two major limitations of EJB are its lack of the notion of user accounts and support for their management (i.e., adding, deleting, (un)assigning to/from roles), as well as the lack of support for user sessions and role activation. According to our analysis, which is summarized in Table 5, this limitation results in two thirds of Core RBAC functions being dependent on vendor-specific extensions (see column “Additional Required Components”). The architects of EJB might have intentionally left the notion of user and support for user management as well as session and role activation beyond the scope of the specification. In order to provide standard support for administering and reviewing user accounts, their roles and their sessions, the corresponding administrative interfaces would need to be added to EJB. However, such a revision would be contrary to the emerging state of practice for application systems.

The notable trend in IT systems design is to “outsource” the functionality for administering user accounts, and in some cases permissions, to single sign-on (SSO) [PM03] solutions

Core RBAC Functions	EJB Data Structures Support	Additional Required Components	
		User Management	Sessions and Role Activation
<b>Administrative Commands</b>			
AddUser		✓	
DeleteUser		✓	
AssignUser		✓	
DeassignUser		✓	
AddRole	✓		
DeleteRole	✓		
GrantPermission	✓		
RevokePermission	✓		
<b>Supporting System Functions</b>			
CreateSession			✓
DeleteSession			✓
AddActiveRole			✓
DropActiveRole			✓
CheckAccess	✓		
<b>Review Functions</b>			
AssignedUsers		✓	
AssignedRoles		✓	
<b>Advanced Review Functions</b>			
RolePermissions	✓		
SessionPermissions			✓
UserPermissions		✓	
SessionRoles			✓
RoleOperationsOnObject	✓		
UserOperationsOnObject		✓	

Table 5: Functions defined by ANSI Core RBAC and their support by EJB data structures

for new applications [Got05] and to identity management (IdM) solutions for existing applications [BS03]. As a result, user accounts, and sometimes permissions, are administered across multiple application instances and types “outside” of the applications themselves. Therefore, an application system can only be successfully evaluated for compliance with ANSI RBAC when the application is considered together with the corresponding SSO or IdM solution. This condition makes evaluation of support for ANSI RBAC prohibitively expensive for systems designed to work in conjunction with multiple SSO or IdM solutions, as the evaluation would have to be performed for every combination of the system and the supporting SSO/IdM. Defining a separate ANSI RBAC profile for SSO/IdM solutions is a possible alternative to explore.

The other limitations of the EJB specification relate to Hierarchical and Constrained RBAC



components of ANSI RBAC. The EJB specification does not define support for either role hierarchies or separation of duty. In Sections 5.1.2 and 5.1.3, we sketch approaches for supporting the two components. However, additional data must be maintained outside of the standard deployment descriptor in order to implement role hierarchies.

## 7 Conclusion

In this paper, we analyzed support for ANSI RBAC by EJB 3.0 compliant systems. Specifically, we defined a configuration of the EJB protection system in precise and unambiguous terms using set theory. Based on this configuration definition, we formally specified the semantics of authorization decisions in EJB. We analyzed support for various ANSI RBAC components in EJB, and illustrated our discussion with an example.

Our analysis shows a mismatch between the access control architectures of EJB and ANSI RBAC. Although, the specification of access controls in EJB does employ roles, it does not fully support even Core ANSI RBAC. The limitations are mainly due to the lack of support of (1) user accounts and their management, (2) user sessions, and (3) role activation. While these limitations can be easily worked around through vendor-specific and implementation dependent extensions, each EJB implementation would have to be evaluated for ANSI RBAC separately. In order to provide standard support for administering and reviewing user accounts, their roles and their sessions, the corresponding administrative interfaces would need to be added to EJB, which would be contrary to the emerging practice of “outsourcing” such functions to enterprise-wide single sign-on and identity management solutions.

This paper establishes a framework for analyzing support for ANSI RBAC in EJB implementations. The results provide directions for EJB developers implementing ANSI RBAC in their systems, and criteria application owners in selecting such EJB implementations that support required and optional components of ANSI RBAC.

**Action Item 3** *Write acknowledgements section*

## References

- [Ahn00] Gail-Joon Ahn. Role-based access control in DCOM. *Journal of Systems Architecture*, 46(13):1175–1184, 2000.
- [ANS04] ANSI. ANSI INCITS 359-2004 for role based access control, 2004.
- [AS01] Gail-Joon Ahn and Ravi Sandhu. Decentralized user group assignment in Windows NT. *The Journal of Systems and Software*, 56(1):39–49, 2001.
- [Awi97] Roland Awischus. Role based access control with security administration manager (SAM). In *The Second ACM Workshop on Role-Based Access Control*, pages 61–68, Fairfax, Virginia, USA, 1997. ACM Press.
- [Bar95] John Barkley. Implementing role-based access control using object technology. In *The First ACM Workshop on Role-Based Access Control*, pages 93–98, Fairfax, Virginia, USA, 1995. ACM Press.
- [Bar97] Larry S. Bartz. hyperDRIVE: leveraging LDAP to implement RBAC on the web. In *Proceedings of the Workshop on Role-based Access Control*, pages 69–74, New York, NY, 1997. ACM Press.
- [BC98] John Barkley and Anthony Cincotta. Managing role/permission relationships using object access types. In *The Third ACM Workshop on Role-Based Access Control*, pages 73–80, Fairfax, Virginia, USA, 1998. ACM Press.
- [BL75] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, MITRE, March 1975.

- [BO06] Vinith Bindiganavale and Jinsong Ouyang. Role based access control in enterprise application—security administration and user management. In *2006 IEEE International Conference on Information Reuse and Integration*, pages 111–116, Waikoloa Village, HI, September 2006. IEEE, IEEE Press.
- [BS03] D.A. Buell and R. Sandhu. Identity management. *IEEE Internet Computing*, 7(6):26–28, Nov.-Dec. 2003.
- [CO02] David W. Chadwick and Alexander Otenko. The PERMIS X.509 role based privilege management infrastructure. In *SACMAT '02: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, pages 135–140, New York, NY, USA, 2002. ACM Press.
- [DK06] Linda DeMichiel and Michael Keith. JSR-220: Enterprise JavaBeans specification, version 3.0: EJB core contracts and requirements. Specification v.3.0 Final Release, Java Community Program, May 2006.
- [DYK01] Linda G. DeMichiel, L. Ümit Yalçınalp, and Sanjeev Krishnan. *Enterprise JavaBeans Specification, Version 2.0*. Sun Microsystems, 2001.
- [Edd99] Guy Eddon. The COM+ security model gets you out of the security programming business. *Microsoft Systems Journal*, 1999(11), 1999.
- [ES95] Jeremy Epstein and Ravi Sandhu. Netware 4 as an example of role-based access control. In *Proceedings of the First ACM Workshop on Role-Based Access Control*, pages 71–82, Gaithersburg, Maryland, USA, 1995. ACM Press.
- [Fad99] Glenn Faden. RBAC in UNIX administration. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 95–101, New York, NY, USA, 1999. ACM Press.
- [FBK99] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):34–64, February 1999.
- [FK92] D. Ferraiolo and R. Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, USA, 1992. National Institute of Standards and Technology/National Computer Security Center.
- [FKK96] Alan O. Freier, Philip Karlton, and Philip Karlton. The ssl protocol version 3.0, 1996.
- [FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.
- [GDS97] Mats Gustafsson, Benoit Deligny, and Nahid Shahmehri. Using NFS to implement role-based access control. In *Proceedings of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pages 299–304, Washington, DC, USA, 1997. IEEE Computer Society.
- [Giu98] Luigi Giuri. Role-based access control in Java. In *Proceedings of the Third ACM Workshop on Role-Based Access Control*, pages 91–99, Fairfax, Virginia, USA, 1998. ACM Press.
- [Giu99] Luigi Giuri. Role-based access control on the Web using Java. In *Proceedings of the Fourth ACM Workshop on Role-based Access Control*, pages 11–18, New York, NY, USA, 1999. ACM Press.
- [Got05] G. Goth. Identity management, access specs are rolling along. *IEEE Internet Computing*, 9(1):9–11, Jan.-Feb. 2005.
- [Gut01] Kurt Gutzmann. Access control and session management in the HTTP environment. *IEEE Internet Computing*, 5(1):26–35, 2001.

- [HFB01] Bret Hartman, Donald J. Flinn, and Konstantin Beznosov. *Enterprise Security With EJB and CORBA*. John Wiley & Sons, Inc., New York, 2001.
- [IET03] IETF. Network file system (NFS) version 4 protocol, 2003.
- [IT00] ITU-T. Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks. ITU-T Standard, ISO/IEC 9594-8, 2000.
- [Lam71] Butler W. Lampson. Protection. In *5th Princeton Conference on Information Sciences and Systems*, page 437, New York, NY, USA, 1971. ACM Press.
- [Mey97] William J. Meyers. RBAC emulation on trusted dg/ux. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, pages 55–60, Fairfax, Virginia, USA, 1997. ACM Press.
- [Not95] LouAnna Notargiacomo. Role-based access control in oracle7 and trusted oracle7. In *the First ACM Workshop on Role-Based Access Control*, pages 65–69, Gaithersburg, Maryland, USA, 1995. ACM Press.
- [Obe00] Robert J. Oberg. *Understanding & programming COM+: a practical guide to Windows 2000 DNA*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
- [OF02] Rafael R. Obelheiro and Joni S. Fraga. Role-based access control for CORBA distributed object systems. In *Proceedings of the The IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, page 53, Washington, DC, USA, 2002. IEEE.
- [OMG99] OMG. The common object request broker: Architecture and specification. Specification formal/99-10-08, Object Management Group, 1999.
- [OMG02] OMG. Common object services specification, security service specification v1.8, 2002.
- [OMG04] OMG. Common object request broker architecture: Core specification v3.0.3, 2004.
- [OMG07a] OMG. Unified Modeling Language: Infrastructure, v2.1.1. OMG document# formal/07-02-06, February 2007.
- [OMG07b] OMG. Unified Modeling Language: Superstructure, v2.1.1. OMG document# formal/07-02-05, February 2007.
- [PM03] Andreas Pashalidis and Chris J. Mitchell. A taxonomy of single sign-on systems. In *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia*, volume 2727 of *Lecture Notes in Computer Science*, pages 249–264. Springer, July 9-11 2003.
- [PSA01] Joon S. Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Transactions on Information and System Security (TISSEC)*, 4(1):37–71, February 2001.
- [RS98] C. Ramaswamy and R. Sandhu. Role-based access control features in commercial database management systems. In *Proc. 21st NIST-NCSC National Information Systems Security Conference*, pages 503–511, Arlington, VA, USA, 1998. National Institute of Standards and Technology/National Computer Security Center.
- [RSB05] Ed Roman, Rima Patel Sriganesh, and Gerald Brose. *Mastering Enterprise JavaBeans*. Wiley Publishing, 10475 Crosspoint Boulevard, Indianapolis, IN 46256, USA, third edition, 2005.
- [SA98] Ravi Sandhu and Gail-Joon Ahn. Decentralized group hierarchies in UNIX: An experiment and lessons learned. In *Proc. 21st NIST-NCSC National Information Systems Security Conference*, pages 486–502, Arlington, Virginia, USA, 1998. National Institute of Standards and Technology/National Computer Security Center.

- [SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- [SCH<sup>+</sup>04] Carla Sadtler, Lee Clifford, Jeff Heyward, Arihiro Iwamoto, Noelle Jakusz, Lars Bek Laursen, WonYoung Lee, Isabelle Mauny, Shafkat Rabbi, and Ascension Sanchez. *IBM WebSphere Application Server V5.1 System Management and Configuration WebSphere Handbook Series*. IBM International Technical Support Organizat, October 2004.
- [SFK00] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Workshop on Role-Based Access Control*, Berlin, 2000. ACM.
- [Sun01] Sun. Java authentication and authorization service (JAAS). <http://java.sun.com/products/jaas/>, 2001.
- [Sun07] Sun. Remote method invocation, 2007.
- [WdSFW<sup>+</sup>02] C. M. Westphall, Joni da Silva Fraga, M. S. Wangham, R. R. Obelheiro, and Lau Cheuk Lung. PoliCap—proposal, development and evaluation of a policy service and capabilities for CORBA Security. In *SEC '02: Proceedings of the IFIP TC11 17th International Conference on Information Security*, pages 263–274, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.
- [WF99] C. Westphall and J. Fraga. A large-scale system authorization scheme proposal integrating Java, CORBA and web security models and a discretionary prototype. In *Latin American Network Operations and Management Symposium*, pages 14–25, Rio de Janeiro, Brazil, December 1999. IEEE Press.
- [WHK97] M. Wahl, T. Howes, and S. Kille. RFC 2251: Lightweight directory access protocol (v3), 1997.
- [Won97] Raymond K. Wong. Rbac support in object-oriented role databases. In *The Second ACM Workshop on Role-Based Access Control*, pages 109–120, Fairfax, Virginia, USA, 1997. ACM Press.
- [ZM04] Wei Zhou and Christoph Meinel. Implement role based access control with attribute certificates. In *The 6th International Conference on Advanced Communication Technology (ICACT2004)*, volume 1, pages 536–541, Korea, Feb 2004. National Computerization Agency, Electronics and Telecommunications Research Institute, Korea.
- [ZSN<sup>+</sup>06] Fangzhou Zhang, Xiaoyu Sheng, Yingying Niu, Fengpin Wang, and Han Zhang. The research and scheme of RBAC using J2EE security mechanisms. In Raj Jain, Benjamin B. Dingel, Shozo Komaki, and Shlomo Ovadia, editors, *Broadband Access Communication Technologies*, volume 6390, page 63900L, Boston, MA, USA, 2006. SPIE.