Analysis of ANSI RBAC Support in EJB

Wesam Darwish, The University of British Columbia, Canada

Konstantin Beznosov, The University of British Columbia, Canada

ABSTRACT

This paper analyzes access control mechanisms of the Enterprise Java Beans (EJB) architecture and defines a configuration of the EJB protection system in a more precise and less ambiguous language than the EJB 3.0 standard. Using this configuration, the authors suggest an algorithm that formally specifies the semantics of authorization decisions in EJB. The level of support is analyzed for the American National Standard Institute's (ANSI) specification of Role-Based Access Control (RBAC) components and functional specification in EJB. The 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 support ANSI RBAC required components. Other vendor-specific extensions are necessary to support ANSI RBAC optional components. Fundamental limitations exist, however, due to the impracticality of some aspects of the ANSI RBAC standard itself. This paper sets up a framework for assessing implementations of ANSI RBAC for EJB systems.

Keywords: *American National Standard Institute, Enterprise Java Beans, Middleware, Role-Based Access Control, Security*

INTRODUCTION

The American National Standard for Information Technology Role-Based Access Control (ANSI RBAC) (ANSI, 2004) 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 commercial enterprises better than lattice-based mandatory access control (MAC) (Bell & LaPadula, 1975) and owner-based discretionary access control (DAC) (Lampson, 1971). 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 constraints (*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 (Notargiacomo, 1995), NetWare (Epstein & Sandhu, 1995), Java (Giuri,

DOI: 10.4018/jsse.2011040102

1998), DG/UX (Meyers, 1997), J2EE (Zhang, Sheng, Niu, Wang, & Zhang, 2006; Bindiganavale & Ouyang, 2006), object-oriented systems (Barkley, 1995), object-oriented databases (Wong, 1997), MS Windows NT (Barkley & Cincotta, 1998), enterprise security management systems (Awischus, 1997). Evidence of RBAC becoming a dominant access control paradigm is the approval of the American National Standard Institute (ANSI) RBAC Standard (ANSI, 2004) in 2004.

At the same time, commercial middleware technologies−such as Common Object Request Broker Architecture (CORBA) (OMG, 1999), COM+ (Oberg, 2000), Enterprise Java Beans (EJB) (DeMichiel, Yalçinalp, & Krishnan, 2001)−matured, with distributed enterprise applications routinely developed with the use of middleware. Each middleware technology, however, comes with its own security subsystem (Eddon, 1999; OMG, 2002; Hartman, Flinn, & Beznosov, 2001), sometimes dependent on and specific to the underlying operating system (OS). For instance, COM+ security (Eddon, 1999) is tied into Microsoft Windows OS and its services.

The ability of a particular middleware technology to support specific types of access control policy is an open and practical question. It is not a simple question for 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 the 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 abstractions directly supported by 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 (RBAC) models as defined by ANSI RBAC Standard (ANSI, 2004).

We have formalized the authorization-related parts of EJB v.3.0 (DeMichiel & Keith, 2006) into a 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 possible, we showed how the corresponding ANSI RBAC construct can be expressed in the language of the EJB protection state. In cases when support for a 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.

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 both required and optional components of ANSI RBAC.

The rest of the paper is organized as follows: In the next section, we provide an overview of ANSI RBAC and EJB. We then discuss related work. The following section formally defines the protection state of the EJB access control subsystem. Then we discuss how an ANSI RBAC based access policy maps to the EJB protection state, and we provide an example. Following that, we discuss the results of our analysis. We present our conclusion in the last section of the paper.

BACKGROUND

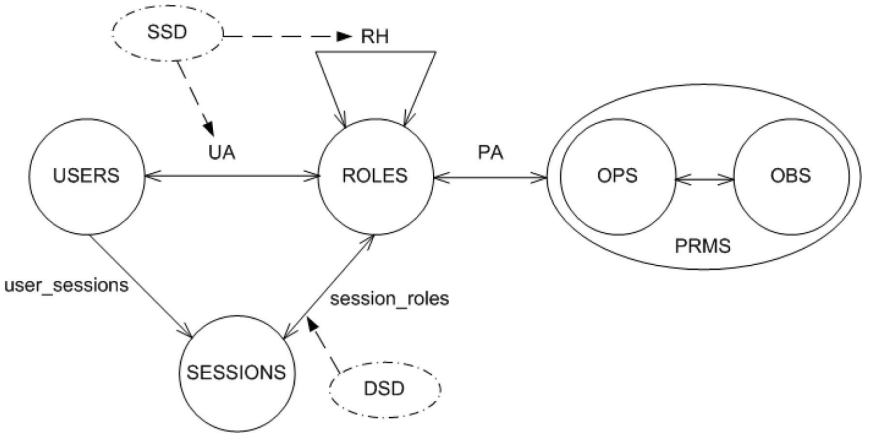
This section provides the background to ANSI RBAC and EJB Security that is necessary in order to understand the rest of the paper. Readers familiar with both can skip directly to the next section.

Overview of ANSI RBAC

Role-Based Access Control (RBAC) was introduced more than a decade ago (Ferraiolo & Kuhn, 1992; Sandhu, Coyne, Feinstein, & Youman, 1996). Over the years, RBAC has enjoyed significant attention. Many research papers have been 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 (Sandhu, Ferraiolo, & Kuhn, 2000). A second version was later publicly released in 2001 (Ferraiolo, Sandhu, Gavrila, Kuhn, & Chandramouli, 2001). 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 submission to the American National Standards Institute (ANSI). The standard was later approved in 2004 (ANSI, 2004). The ANSI RBAC standard consists of two main parts, as described in the following sections.

Figure 1. ANSI RBAC sets, relations, and main functions



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. Figure 1 depicts these RBAC components.

Core RBAC

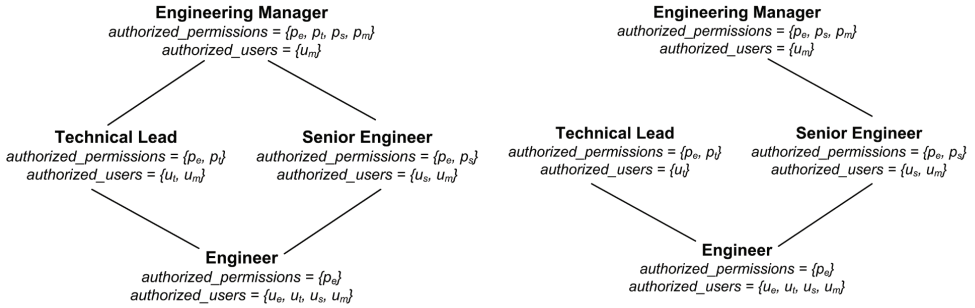
Core RBAC defines the minimum set of elements required to achieve RBAC functionality. At a minimum, core RBAC must be implemented in RBAC systems. The other components described below, which are independent of each other, can be implemented separately.

Core RBAC elements are defined as follows (ANSI, 2004, 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 $(r) = \{u \in USERS \,|\, (u,r) \in UA\}$.
- $PRMS = 2^{(OPS \times OBS)}$, 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 $(r) = \{p \in PRMS \,|\, (p,r) \in PA\}$.
- $Op(p : PRMS) \rightarrow \{op \in OPS\}$, the permission *to operation mapping, which gives the set of operations associated with permission p.*
- $Ob(p : PRMS) \rightarrow \{ob \in OBS\}$, 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 $(s_i) \subseteq \{r \in ROLES \,|\, (session\_users(s_i), r) \in UA\}$.
- *avail_session_perms* $(s : SESSIONS) \rightarrow 2^{PRMS}$, the per-

Figure 2. Examples of Hierarchical RBAC



missions 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 2, 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 the multiple inheritances 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 is limited to only one descendant.

In the presence of role hierarchy, the following is defined, where $r_{senior} \geq r_{junior}$ indicates that r_{senior} inherits all permissions of r_{junior}, and all users of r_{senior} are also users of r_{junior}:

- *authorized_users* $(r) = \{u \in USERS \,|\, r' \geq r, (u, r') \in UA\}$ is the mapping of role r onto a set of users.
- *authorized_permissions* $(r) = \{p \in PRMS \,|\, r \geq r', (p, r') \in PA\}$ is the mapping of role r onto a set of permissions.

Constrained RBAC

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

Functional Specification

For the four components defined in the RBAC reference model, the RBAC System and Administrative Functional Specification define 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 the operations required for the creation and maintenance of RBAC element sets and relations. Examples of these operations are listed

below. A complete list of these operations, as well as their formal definitions 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 the 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.

Overview of EJB Security

In this section we provide an overview of EJB architecture, the main components of an EJB system, as well as the declarative and runtime aspects of EJB systems.

EJB

This section provides a brief and informal overview of Enterprise Java Beans (EJB). More information can be found in the corresponding EJB specification. Readers familiar with EJB are advised to proceed to the EJB Security Subsystem section.

The EJB standard (DeMichiel & Keith, 2006) defines an architecture for developing and deploying server-side components written in Java programming language. 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.

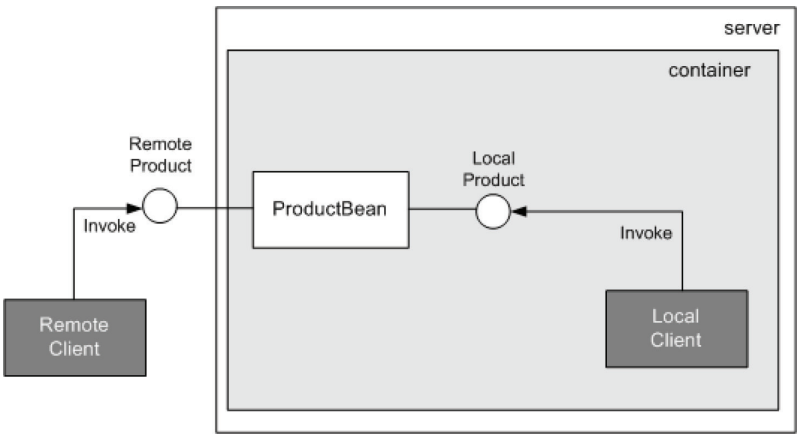
EJB architecture, similar to other middleware technologies, allows application developers to implement their business logic without having to handle transactions, state management, multi-threading, connection pooling, and other platform-dependent deployment issues.

EJB architecture consists of the following basic parts. These parts are also shown in Figure 3 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 3 as ProductBean. The EJB specification (DeMichiel & Keith, 2006) 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

Figure 3. Basic parts of EJB architecture for an example Enterprise Java Bean Product



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 the runtime environment to one or more containers. Since EJB specification does not explicitly define the separation of roles between containers and servers, they are usually inseparable and come 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. (DeMichiel & Keith, 2006) The EJB container tools handle the generation of the required logic in order to support remote access to methods defined by this interface.

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 Java programming language, fully compliant EJB deployment environments support the Internet Inter-ORB Protocol (IIOP) (OMG, 2004), leveraging IIOP and the Common Secure Interoperability Protocol Version 2 (CSIv2) (OMG, 2004) capabilities, which allow CORBA clients to access enterprise bean objects, and which can be written in languages other than Java.

Declarative Part Defining remote and local interfaces as well as implementing the business logic in EJB is as easy as in standard Java. Figure 4 shows an example of an enterprise bean remote interface definition, and Figure 5 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

Figure 4. Defining a remote interface for the Product enterprise bean (Product.java)

```
import javax.ejb.Remote;

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

Figure 5. Implementing the remote interface for the Product enterprise bean (ProductBean.java)

```
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;
    }
}
```

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*, which play specific roles in the life cycle of enterprise beans predefined by the EJB specification.

Runtime Part While the remote object model for EJB components is based on the Remote Method Invocation (RMI) API (ORACLE, 2007), 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 a more detailed explanation of EJB technology, please refer to Roman, Sriganesh, and Brose (2005).

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 of 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 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 about 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 the EJB specification focus largely on authentication and access control. The specification relies on CSIv2 for message protection, and it leaves support for security auditing to the discretion of container vendors. We describe the EJB access control architecture later.

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 a local authentication service, which is not predefined by the specification. Once the container authenticates the client (or verified their credentials), it enforces access control policies. The notion of a *principal* is used in the EJB specification to refer to authenticated clients.

Administration Some of the security administration tasks of EJB servers are performed through changes in deployment descriptors. This includes the definition of security roles, method permissions, and the 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.

RELATED WORK

Over the past decade, there has been no shortage of papers proposing ways to support RBAC.

Most of this work, however, is about support for RBAC96 (Sandhu et al., 1996), which defines the reference models for plain, hierarchical, and constrained RBAC, but does not specify the functions to be supported by an RBAC implementation. The paucity of analysis or proposals for supporting ANSI RBAC is not surprising, given the fact that the standard was published in 2004. Because of the lack of research on support for ANSI RBAC, and because of the significant similarities between RBAC96 and ANSI RBAC, we review related work on supporting or implementing RBAC96 in operating systems, databases, web applications, and distributed systems, including middleware. Since the mainstream operating systems, with the exception of Solaris (Sun Microsystems Inc., 2000), do not provide direct support for RBAC, researchers and developers have been employing either groups (e.g., Sandhu & Ahn, 1998; Ahn & Sandhu, 2001), or user accounts (e.g., Faden, 1999; Chalfant, 2003) to simulate roles. This choice determines whether more than one role can be activated in a session. Role hierarchies are either not supported (Faden, 1999; Sun Microsystems Inc., 2000), or are simulated by maintaining additional system files with the role hierarchy and various bookkeeping data (Sandhu & Ahn, 1998; Ahn & Sandhu, 2001). None of the implementations we reviewed support static SoD. Just one case of dynamic SoD comes as a side-effect with those implementations that simulate roles with user accounts (Faden, 1999; Chalfant, 2003): the role set in this DSoD is equal to the set of all roles in the system, and the cardinality of the role set is exactly one. In other words, any session can have only one role activated at any given time; the current role is deactivated when another role is activated.

We analyzed DB2 (Tran & Mohan, 2006) and MySQL (MySQL AB, 2007), and updated the analysis of RBAC support in commercial database management systems (DBMS)−conducted by Ramaswamy and Sandhu (1998)− with the latest versions of the corresponding systems. Commercial DBMS continue to have the most advanced support for RBAC96. Informix Dynamic Server v7.2 (IBM, 2005), IBM DB2 (Tran & Mohan, 2006), Sybase Adaptive Server v11.5 (Sybase Inc., 2005), and Oracle Enterprise Server v8.0 (Baylis, Lane, & Lorentz, 2003) directly support roles and role hierarchies. Only Oracle and Sybase allow users to have more than one role activated at any time, though. On the other hand, Informix also provides limited support for dynamic SoD, and Sybase features support for both types of SoD.

In RBAC implementations for client-server systems, including Web applications, roles are either "pushed" from the client to the server in the form of attribute certificates or HTTP cookies (Gutzmann, 2001; Park, Sandhu, & Ahn, 2001; Robles, Choi, Yeo, & Kim, 2008), or "pulled" by the server from a local or remote database (Bartz, 1997; Ferraiolo, Barkley, & Kuhn, 1999; Park et al., 2001; Chadwick & Otenko, 2002; Zhou & Meinel, 2004). The former enables selective activation of roles by users, and the latter simplifies the implementation of client authentication, but activates all of the assigned roles for the user. However, Web implementation of NIST RBAC (Ferraiolo et al., 1999) has a hybrid design, which allows the user to select the roles to be "pulled" by the server. A number of implementations use a database, possibly accessible through the Lightweight Directory Access Protocol (LDAP) (Wahl, Howes, & Kille, 1997) front-end to store role and other information (Bartz, 1997; Gutzmann, 2001; Park et al., 2001; Zhou & Meinel, 2004). Role hierarchies are only supported by some implementations, using either manual assignment of permissions of junior roles to senior ones (Park et al., 2001), additional files (Giuri, 1999), a database (Ferraiolo et al., 1999) or an LDAP server (Chadwick & Otenko, 2002; Zhou & Meinel, 2004). JRBAC-WEB (Giuri, 1999) and RBAC/Web (Ferraiolo et al., 1999) also support both types of SoD.

The work most relevant to ours addresses support for RBAC in middleware. Ahn (2000) outlines a proposal for enforcing RBAC policies for distributed applications that utilize Microsoft's Distributed Component Object Model (DCOM) (Brown & Kindel, 1998; Microsoft,

1998). His proposal employs the following elements of Windows NT's architecture: (1) a registry for storing and maintaining the role hierarchy and permission-to-role assignment (PA); (2) user groups for simulating roles and maintaining user-to-role assignment (UA); and (3) a custom built security provider that follows the RBAC model to make access control decisions, which are requested and enforced by the DCOM run-time. Since the support for role hierarchy is indicated, but not explained, by Ahn (2000), we assume that the Windows NT registry can be used to encode the hierarchy so that the RBAC security provider can refer to it while making authorization decisions. Similar to the proposals for RBAC support in operating systems, the use of OS user groups for simulating roles enables activation of more than one role. Yet, as with the pull model in client-server systems, all assigned roles are activated, leaving no choice for the user. Ahn (2000) does not indicate support for any kind of SoD, nor does he explain how RBAC policies can be enforced consistently and automatically in a multi-computer deployment of DCOM-accessible objects.

RBAC-JaCoWeb (Westphall & da Silva Fraga, 1999; Obelheiro & da Silva Fraga, 2002) utilizes the PoliCap (Westphall, da Silva Fraga, Wangham, Obelheiro, & Lung, 2002) policy server to implement CORBASec specification in a way that supports RBAC. PoliCap holds all data concerning security policies within a CORBASec policy domain, including users, roles, user-to-role and role-to-permission assignments, role hierarchy relations, and SoD constraints. Most of the authorization policy enforcement is performed by an RBAC-JaCoWeb CORBA security interceptor. At the time of the client binding to a CORBA object, the interceptor obtains necessary data from the PoliCap server and instantiates CORBASec-compliant DomainAccessPolicy and RequiredRights objects containing the privilege and control attributes appropriate for the application object. When the client makes invocation requests later, the access decisions are then performed based on the local instances of these objects. Initially, the client security credentials object−created as part of the binding−has no privilege attributes, only AccessId, which is obtained from the client's X.509 certificate used in the underlying SSL connection. If the invocation cannot be authorized with the current set of client privilege attributes, the interceptor "pulls" additional role attributes from the PoliCap server. Only those roles that are (1) assigned to the user, (2) necessary for the invocation in question to be authorized, and (3) not in conflict with any DSoD constraints are activated. These role attributes are added to the client's credentials and are later re-used on the server for other requests from the same principal. The extent to which RBAC-JaCoWeb conforms to the CORBASec specification is unclear (Westphall & da Silva Fraga, 1999; Obelheiro & da Silva Fraga, 2002). Nevertheless, RBAC-JaCoWeb serves as an example of implementation-specific extensions to CORBAsec that enable better support for RBAC advanced features, such as role hierarchies and SoD, which−as will be seen from the results of our analysis−cannot be supported without extending a CORBASec implementation with additional operations.

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.

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 an 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 programmatic security, the application developers employ methods called IsCallerInRole and getCallerPrincipal to obtain the information about the caller in order to enforce those access control policies, which cannot be expressed using the declarative approach.

Authorization to invoke the 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 the bean's metadata annotations. Since the 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, who 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" (DeMichiel & Keith, 2006, p. 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 6 uses Unified Modeling Language (UML) (OMG, 2007a, 2007b) 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.

The assignment of permissions to roles is done in method-permission sections. Such sections list roles permitted to invoke one or more methods. When the special role name is "unchecked" it can be used to indicate that all the 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. The special method name "*" refers to all methods on a given bean.

An example of an assignment done through method-permission sections is shown in Table 1. The first row illustrates an assignment of a permission to invoke method m_1 on bean b_1 ($b_1.m_1$) 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 ($b_1.m_2$ and $b_1.m_4$). This means that any principal that has any of these two roles can invoke any of these two methods. The last row provides an example of using "unchecked" and "*" keywords. It states that any principal can invoke method $b_2.m_1$ 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 1, then a principal with role r_2 is granted permission to invoke methods $b_1.m_2$, $b_1.m_4$, $b_2.m_1$, 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 principal−according to Section 17.3.2.3 of the EJB specification (DeMichiel & Keith, 2006), methods with unspecified permissions must be treated by the container as "unchecked." For instance, if $b_1.m_3$ is such a method then any principal would be able to invoke it.

Even though the syntax of the method-permission section allows the listing of more

Figure 6. Relationships among the sections of deployment descriptor used for expressing access control policy and the elements of an EJB application

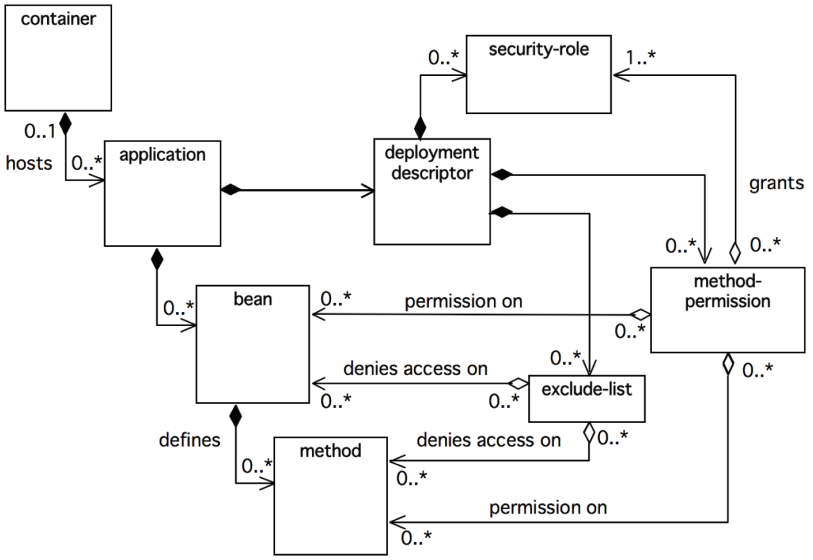


Table 1. Examples of method-permission sections of EJB deployment descriptor. For the sake of clarity, the data representation is converted from XML notation to human-understandable form, with each row corresponding to an individual section

Roles	Methods
r_1	$b_1.m_1$
r_1, r_2	$b_1.m_2$, $b_1.m_4$
"unchecked"	$b_2.m_1$, $b_3.*$

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 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 the following section.

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.

Formalization of the Protection State

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

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

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

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 the deployment descriptor provided with the EJB application. These roles are defined using the security-role tags. This set also includes the special role "unchecked".*
- *B is a set of enterprise beans listed in the enterprise-beans section of the deployment descriptor.*
- *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,* $OPS = \{m_1, m_2, ...\} \cup \{*\}$.
- $M \subseteq B \times OPS$ is the set of *available uniquely identifiable methods. Members of this set are denoted* $b_i.m_j$.
- $MP \subseteq R \times M$ 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 o*f 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 EJB support for RBAC in the following section, 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 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.

ANALYSIS OF SUPPORT FOR ANSI RBAC

As described in the section titled Overview of ANSI RBAC, the ANSI RBAC Reference

Algorithm 1. Authorization decision in EJB. Decide authorization for principal invoking method m_j on bean b_i, where $r_1, r_2, ...r_n \in R$, and $b_i.m_j \in M$

```
Authorize(p : 2^R, b_i.m_j : M) → {allow, deny}
if b_i.m_j ∈ X then
   return deny
end if
for all r ∈ p ∪ {"unchecked"} do
   if (r, b_i.m_j) ∈ MP ∨ (r, b_i*) ∈ MP then
      return allow
   end if
end for
return deny
```

Model defines four major components. In order for a system to conform to ANSI RBAC, Core RBAC must be implemented at a minimum. An ANSI compliant RBAC system can also implement Hierarchical RBAC, which defines hierarchies of roles in addition to everything Core RBAC does. The other two optional components of the standard, Static Separation of Duty (SSD) and Dynamic Separation of Duty (DSD), define relations among roles with respect to user assignments as well as role activation in user sessions.

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. We then provide an example that illustrates the abilities of an EJB system to support ANSI RBAC. Following that, we analyze the degree to which the structures defined in EJB specification support the functional specification of ANSI RBAC. In the Discussion section, we then discuss the results of our analysis.

Reference Model

Core RBAC

Various Core RBAC data elements are mapped readily into EJB using the sets defined in the EJB Protection State section. 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 per*missions to invoke EJB methods provided that these methods do not exist in the exclusion set X. The existence of* $b_i.m_j$,

or b_i. in PRMS rovides 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.
- $Op(p : PRMS) \rightarrow \{op \in OPS\}$, a function t*hat returns a set of operations that are associated with the given permission p.*
- $Ob(p : PRMS) \rightarrow \{ob \in OBS\}$, a function t*hat 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 (DeMichiel & Keith, 2006) does not mandate how users must be supported in an EJB system, various implementations of EJB servers and containers implement extensions to deployment descriptors. 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.

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*. 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 senior roles. Finally, the administrative tools must also keep track of whether permission has been directly assigned to a role, or if 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 Ahn and Sandhu (2001) and Sandhu and Ahn (1998) 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 earlier−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 (Ferraiolo et al., 1999) for Common Gateway Interface (CGI) based Web applications, and in (Giuri, 1999) for Java Authentication and Authorization Service (JAAS) (ORACLE, 2001) 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.

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.

Example

In this section, we present an example that illustrates the abilities of an EJB system to support ANSI RBAC. As discussed in the previous section, 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 7. 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 8, users are allowed to invoke various methods on a specific EJB. These roles are defined as follows:

- *Employee* represents a company employee.

Figure 7. Example EngineeringProject and Employee session beans

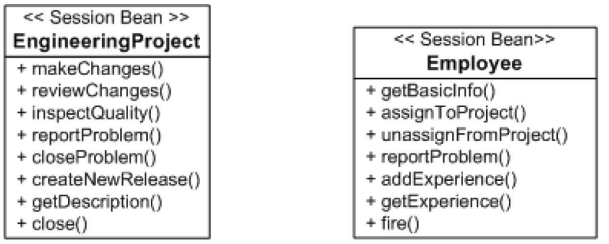


Figure 8. Authorization policy for the example EJB system describing what actions are allowed. All other actions are denied.

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 to which they are assigned.
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 product 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.

- *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.

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 9 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 = { Employee, Engineering Department, Engineer, Product Engineer, Quality Engineer, Project Lead, Director}
- B = { EngineeringProject, Employee }
- OPS = { makeChanges, reviewChanges, inspectQuality, reportProblem, closeProblem, createNewRelease, getDescription, close, getBasicInfo, assignToProject, unassignFromProject, addExperience, getExperience, fire}
- M = { EngineeringProject.makeChanges, EngineeringProject.reviewChanges, EngineeringProject.inspectQuality, Engineer-

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

Roles	**Methods**														
	EngineeringProject Bean							*Employee Bean*							
	make-Changes()	review-Changes()	inspec-tQuality()	reportProb-lem()	close Problem()	createNew Release()	getDescrip-tion()	close()	getBasicIn-fo()	assignTo-Project()	unas-sign-From-Pro-ject()	addEx-peri-ence()	getExpe-rience()	fire()
Employee									√				√	
Engineering Department				√			√		√				√	
Engineer	√	√							√				√	
Product Engineer						√			√				√	
Quality Engineer			√						√				√	
Project Lead					√				√				√	
Director								√	√	√	√	√	√	√

Table 4. Example users, groups, and group membership

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

Figure 9. Example EJB system role mappings

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 assignment

ingProject.reportProblem, EngineeringProject.closeProblem, EngineeringProject.createNewRelease, EngineeringProject.getDescription, EngineeringProject.close, Employee.getBasicInfo, Employee.assignToProject, Employee.unassignFromProject, Employee.addExperience, Employee.getExperience, Employee.fire }

- *MP* = {(Employee, Employee.getBasicInfo),
(Employee, Employee.getExperience),
(Engineering Department, EngineeringProject.reportProblem),
(Engineering Department, EngineeringProject.getDescription),
(Engineering Department, Employee.getBasicInfo),
(Engineering Department, Employee.getExperience),
(Engineer, EngineeringProject.makeChanges),
(Engineer, EngineeringProject.reviewChanges),
(Engineer, Employee.getBasicInfo),
(Engineer, Employee.getExperience),
(Product Engineer, EngineeringProject.createNewRelease),
(Engineering Department, Employee.getBasicInfo),
(Engineering Department, Employee.getExperience),
(Quality Engineer, EngineeringProject.inspectQuality),
(Quality Engineer, Employee.getBasicInfo),
(Quality Engineer, Employee.getExperience),
(Project Lead, EngineeringProject.closeProblem),
(Project Lead, Employee.getBasicInfo),
(Project Lead, Employee.getExperience),
(Director, EngineeringProject.close),
(Director, Employee.getBasicInfo),
(Director, Employee.assignToProject),
(Director, Employee.unassignFromProject),
(Director, Employee.addExperience),
(Director, Employee.getExperience),
(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 *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, management }
- *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.

Functional Specification

This section reports on the results of our analysis of the support that the EJB standard (DeMichiel & Keith, 2006) 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 the ANSI publication (2004) 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 needed 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 (Sadtler et al., 2004) 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 (Wahlet al., 1997) 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 *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 supports 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.

Discussion

The results of our analysis suggest that 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

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

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

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) (Pashalidis & Mitchell, 2003) solutions for new applications (Goth, 2005) and to identity management (IdM) solutions for existing applications (Buell & Sandhu, 2003). 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. 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.

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.

To support this rising trend, it is possible to explore extending the ANSI RBAC standard, as well as the EJB standard to define profiles for supporting SSO/IdM solutions. This would also require exploring options for providing proper support for role activation and deactivation in order to adhere to the principle of least privilege. Clearly, activating the roles assigned to a user all at the same time violates this principle. On the other hand, allowing one role to be active at a time would not provide proper role activation support because the user may need the permissions assigned to more than one role in order to invoke a certain operation, in the absence of role hierarchy, for example. Other issues to explore include whether role activation should occur upon user authentication or upon method invocation, when roles should be deactivated, and whether roles should be activated with or without user intervention.

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.

REFERENCES

Ahn, G.-J. (2000). Role-based access control in DCOM. *Journal of Systems Architecture*, *46*(13), 1175–1184. doi:10.1016/S1383-7621(00)00017-5

Ahn, G.-J., & Sandhu, R. (2001). Decentralized user group assignment in Windows NT. *Journal of Systems and Software*, *56*(1), 39–49. doi:10.1016/S0164-1212(00)00084-4

ANSI. (2004). *ANSI INCITS 359-2004 for role based access control*. Retrieved from http://intelligrid.ipower.com/IntelliGrid_Architecture/New_Technologies/Tech_ANSI_INCITS_359-2004_Role_Based_Access_Control_(RBAC).htm

Awischus, R. (1997). Role based access control with security administration manager (SAM). In *Proceedings of the Second ACM Workshop on Role-Based Access Control* (pp. 61-68). New York, NY: ACM Press.

Barkley, J. (1995). Implementing role-based access control using object technology. In *Proceedings of the First ACM Workshop on Role-Based Access Control* (pp. 93-98). New York, NY: ACM Press.

Barkley, J., & Cincotta, A. (1998). Managing role/permission relationships using object access types. In *Proceedings of the Third ACM Workshop on Role-Based Access Control* (pp. 73-80). New York, NY: ACM Press.

Bartz, L. S. (1997). hyperDRIVE: Leveraging LDAP to implement RBAC on the web. In *Proceedings of the ACM Workshop on Role-Based Access Control* (pp. 69-74). New York, NY: ACM Press.

Baylis, R., Lane, P., & Lorentz, D. (2003). *Oracle database administrator's guide*. Retrieved from http://otn.oracle.com/pls/db10g/db10g.homepage

Bell, D. E., & LaPadula, L. J. (1975). *Secure computer systems: Unified exposition and multics interpretation* (Technical Report No. ESD-TR-75-306). Bedford, MA: MITRE.

Bindiganavale, V., & Ouyang, J. (2006, September). Role based access control in enterprise application-security administration and user management. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, Waikoloa Village, HI (pp. 111-116). Washington, DC: IEEE Computer Society.

Brown, N., & Kindel, C. (1998). *Distributed component object model protocol-DCOM/1.0*. Retrieved from http://www.ietf.org/proceedings/43/I-D/draft-brown-dcom-v1-spec-03.txt

Buell, D., & Sandhu, R. (2003). Identity management. *IEEE Internet Computing*, *7*(6), 26–28. doi:10.1109/MIC.2003.1250580

Chadwick, D. W., & Otenko, A. (2002). The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies* (pp. 135-140). New York, NY: ACM Press.

Chalfant, T. M. (2003). *Role based access control and secure shell - a closer look at two Solaris™ operating environment security features*. Redwood Shores, CA: Sun BluePrints™ OnLine.

DeMichiel, L. G., & Keith, M. (2006). *JSR-220: Enterprise JavaBeans 24 specification, version 3.0: EJB core contracts and requirements (Specification No. v.3.0 Final Release)*. Retrieved from http://jcp.org/aboutJava/communityprocess/pfd/jsr220/index.html

DeMichiel, L. G., Yalçinalp, L. Ü., & Krishnan, S. (2001). *Enterprise JavaBeans specification, version 2.0*. Retrieved from http://java.sun.com/products/ejb/docs.html

Eddon, G. (1999). The COM+ security model gets you out of the security programming business. *Microsoft Systems Journal, 1999*(11).

Epstein, J., & Sandhu, R. (1995). Netware 4 as an example of role-based access control. In *Proceedings of the First ACM Workshop on Role-Based Access Control* (pp. 71-82). New York, NY: ACM Press.

Faden, G. (1999). RBAC in UNIX administration. In *Proceedings of the Fourth ACM Workshop on Role-Based Access Control* (pp. 95-101). New York, NY: ACM Press.

Ferraiolo, D. F., Barkley, J. F., & Kuhn, D. R. (1999). A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, *2*(1), 34–64. doi:10.1145/300830.300834

Ferraiolo, D. F., & Kuhn, R. (1992). Role-based access controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, Baltimore, MD (pp. 554-563).

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, *4*(3), 224–274. doi:10.1145/501978.501980

Giuri, L. (1998). Role-based access control in Java. In *Proceedings of the Third ACM Workshop on Role-Based Access Control*, Fairfax, VA (pp. 91-99). New York, NY: ACM Press.

Giuri, L. (1999). Role-based access control on the Web using Java. In *Proceedings of the Fourth ACM Workshop on Role-Based Access Control* (pp. 11-18). New York, NY: ACM Press.

Goth, G. (2005). Identity management, access specs are rolling along. *IEEE Internet Computing*, *9*(1), 9–11. doi:10.1109/MIC.2005.16

Gutzmann, K. (2001). Access control and session management in the HTTP environment. *IEEE Internet Computing*, *5*(1), 26–35. doi:10.1109/4236.895139

Hartman, B., Flinn, D. J., & Beznosov, K. (2001). *Enterprise security with EJB and CORBA*. New York, NY: John Wiley & Sons.

IBM. (2005). *IBM informix dynamic server administrator's guide*. Retrieved from http://www-306.ibm.com/software/data/informix/pubs/library/ids100.html

Lampson, B. W. (1971). Protection. In *Proceedings of the Fifth Princeton Conference on Information Sciences and Systems* (p. 437).

Meyers, W. J. (1997). RBAC emulation on trusted dg/ux. In *Proceedings of the Second ACM Workshop on Role-Based Access Control* (pp. 55-60). New York, NY: ACM Press.

Microsoft. (1998). *DCOM architecture*. Retrieved from http://www.microsoft.com/NTServer/

MySQL AB. (2007). *MySQL*. Retrieved from http://www.mysql.com

Notargiacomo, L. (1995). Role-based access control in oracle7 and trusted oracle7. In *Proceedings of the First ACM Workshop on Role-Based Access Control* (pp. 65-69). New York, NY: ACM Press.

Obelheiro, R. R., & Fraga, J. S. (2002). Role-based access control for CORBA distributed object systems. In *Proceedings of the IEEE International Workshop on Object-Oriented Real-Time Dependable Systems* (p. 53). Washington, DC: IEEE Computer Society.

Oberg, R. J. (2000). *Understanding & programming COM+: A practical guide to Windows 2000 DNA*. Upper Saddle River, NJ: Prentice Hall.

OMG. (1999). *The common object request broker: Architecture and specification*. Needham, MA: Object Management Group.

OMG. (2002). *Common object services specification, security service specification v1.8*. Needham, MA: Object Management Group.

OMG. (2004). *Common object request broker architecture: Core specification v3.0.3*. Needham, MA: Object Management Group.

OMG. (2007a, February). *Unified modeling language: Infrastructure, v2.1.1*. Needham, MA: Object Management Group.

OMG. (2007b, February). *Unified modeling language: Superstructure, v2.1.1*. Needham, MA: Object Management Group.

ORACLE. (2001). *Java authentication and authorization service (JAAS)*. Retrieved from http://java.sun.com/products/jaas/

ORACLE. (2007). *Remote method invocation*. Retrieved from http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

Park, J. S., Sandhu, R., & Ahn, G.-J. (2001). Role-based access control on the web. *ACM Transactions on Information and System Security*, *4*(1), 37–71. doi:10.1145/383775.383777

Pashalidis, A., & Mitchell, C. J. (2003, July 9-11). A taxonomy of single sign-on systems. In R. Safavi-Naini & J. Seberry (Ed.), *Proceedings of the Eighth Australasian Conference Information Security and Privacy*, Wollongong, Australia (LNCS 2727, pp. 249-264).

Ramaswamy, C., & Sandhu, R. (1998). Role-based access control features in commercial database management systems. In *Proceedings of the 21st NIST-NCSC National Information Systems Security Conference* (pp. 503-511).

Robles, R., Choi, M.-K., Yeo, S.-S., & Kim, T. Hoon. (2008, October). Application of role-based access control for web environment. In *Proceedings of the International Symposium on Ubiquitous Multimedia Computing* (pp. 171-174). Washington, DC: IEEE Computer Society.

Roman, E., Sriganesh, R. P., & Brose, G. (2005). *Mastering enterprise javabeans* (3rd ed.). Indianapolis, IN: Wiley.

Sadtler, C., Clifford, L., Heyward, J., Iwamoto, A., Jakusz, N., & Laursen, L. B. (2004). *IBM websphere application server v5.1 system management and configuration websphere handbook series*. Armonk, NY: IBM International Technical Support Organization.

Sandhu, R., & Ahn, G.-J. (1998). Decentralized group hierarchies in UNIX: An experiment and lessons learned. In *Proceedings of the 21st NIST-NCSC National Information Systems Security Conference* (pp. 486-502).

Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996). Role-based access control models. *IEEE Computer*, *29*(2), 38–47.

Sandhu, R., Ferraiolo, D., & Kuhn, R. (2000). The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control* (pp. 47-63). Application of role-based access control for web environment.

Sun Microsystems Inc. (2000). *RBAC in the Solaris™ operating environment*. Retrieved from http://www.sun.com/software/whitepapers/wp-rbac/wp-rbac.pdf

Sybase Inc. (2005). *System administration guide: Volume 1 - Adaptive server enterprise 15.0*. Retrieved from http://infocenter.sybase.com/help/topic/com.sybase.help.ase_15.0.sag1/sag1.pdf

Tran, S., & Mohan, M. (2006). *Security information management challenges and solutions*. Retrieved from http://www.ibm.com/developerworks/db2/library/techarticle/dm-0607tran/index.html

Wahl, M., Howes, T., & Kille, S. (1997). *RFC 2251: Lightweight directory access protocol (v3)*. Retrieved from http://www.ietf.org/rfc/rfc2251.txt

Westphall, C. M., & da Silva Fraga, J. (1999, December). A large-scale system authorization scheme proposal integrating Java, CORBA and web security models and a discretionary prototype. In *Proceedings of the Latin American Network Operations and Management Symposium*, Rio de Janeiro, Brazil (pp. 14-25). Washington, DC: IEEE Computer Society.

Westphall, C. M., da Silva Fraga, J., Wangham, M. S., Obelheiro, R. R., & Lung, L. C. (2002). PoliCap - proposal, development and evaluation of a policy service and capabilities for CORBA security. In *Proceedings of the IFIP TC11 17th International Conference on Information Security* (pp. 263-274).

Wong, R. K. (1997). RBAC support in object-oriented role databases. In *Proceedings of the Second ACM Workshop on Role-Based Access Control* (pp. 109-120). PoliCap - proposal, development and evaluation of a policy service and capabilities for CORBA security.

Zhang, F., Sheng, X., Niu, Y., Wang, F., & Zhang, H. (2006). The research and scheme of RBAC using J2EE security mechanisms. In *Proceedings of the SPIE Conference on Broadband Access Communication Technologies*, *6390*, 63900L.

Zhou, W., & Meinel, C. (2004, Feb). Implement role based access control with attribute certificates. In *Proceedings of the 6th International Conference on Advanced Communication Technology* (Vol. 1, pp. 536-541). Washington, DC: IEEE Computer Society.

Wesam Darwish is a software architect with AdvancedIO Systems, Inc. He obtained his Master of Applied Science degree in Electrical and Computer Engineering from the University of British Columbia (UBC) in 2009. He was a member of the Laboratory for Education and Research in Secure Systems Engineering (LERSSE) under the supervision of Professor Konstantin Beznosov. His research interests include distributed systems security, software architecture, and access control architectures.

Konstantin (Kosta) Beznosov is an Associate Professor at the Department of Electrical and Computer Engineering, University of British Columbia, where he directs the Laboratory for Education and Research in Secure Systems Engineering. His research interests are usable security, distributed systems security, secure software engineering, and access control. Prior UBC, he was a Security Architect at Hitachi Computer Products (America) and Concept Five. Besides many academic papers on security engineering in distributed systems, he is also a co-author of "Enterprise Security with EJB and CORBA" and "Mastering Web Services Security" books, as well as XACML and several CORBA security specifications. He has served on program committees and/or helped to organize SOUPS, CCS, NSPW, NDSS, ACSAC, SACMAT, CHIMIT. Prof. Beznosov is an associate editor of ACM Transactions on Information and System Security *(TISSEC) and* International Journal of Secure Software Engineering *(IJSSE).*