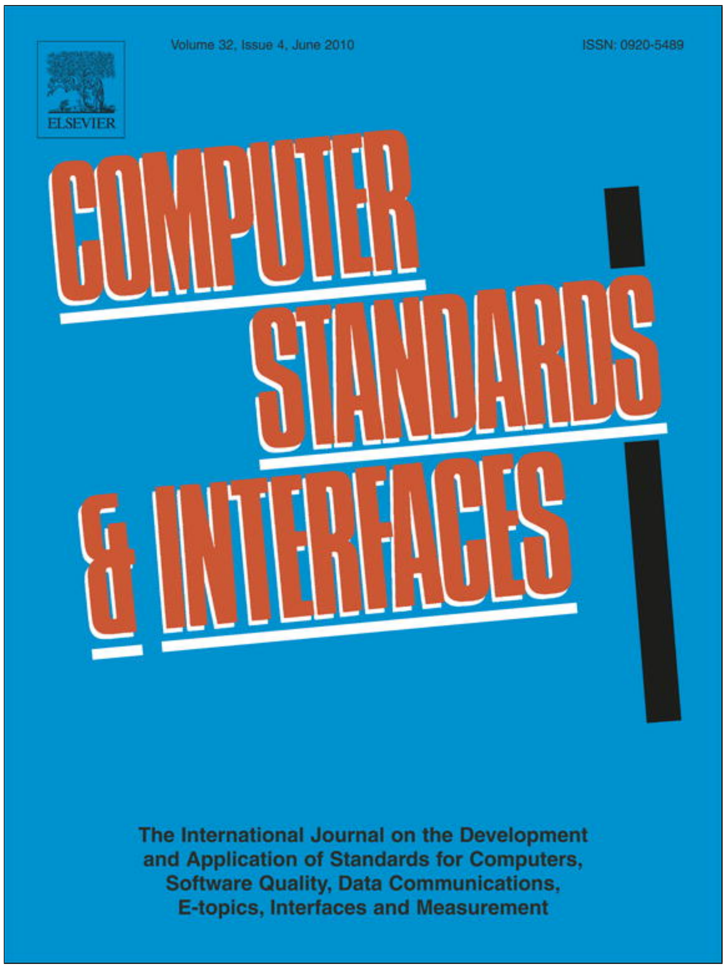
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.


Volume 32, Issue 4, June 2010
ISSN: 0920-5489
ELSEVIER
COMPUTER STANDARDS & INTERFACES
The International Journal on the Development and Application of Standards for Computers, Software Quality, Data Communications, E-topics, Interfaces and Measurement

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Computer Standards & Interfaces 32 (2010) 197–214

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Analysis of ANSI RBAC Support in COM+

Wesam Darwish, Konstantin Beznosov [*]

University of British Columbia, Vancouver, Canada

ARTICLE INFO

Article history:
Received 25 March 2009
Received in revised form 5 October 2009
Accepted 31 December 2009
Available online 25 January 2010

Keywords:
COM+
RBAC
Access control

ABSTRACT

We analyze access control mechanisms of the COM+ architecture and define a configuration of the COM+ protection system in more precise and less ambiguous language than the COM+ documentation. Using this configuration, we suggest an algorithm that formally specifies the semantics of authorization decisions in COM+. 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 COM+.
Our results indicate that COM+ falls short of supporting even Core RBAC. The main limitations exist due to the tight integration of the COM+ architecture with the underlying operating system, which prevents support for session management and role activation, as specified in ANSI RBAC.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

In role-based access control (RBAC) systems, permissions are associated with roles and users are assigned to appropriate roles. A role can represent competency, authority, responsibility or specific duty assignments. A major purpose of RBAC is to facilitate access control administration and review. It arguably addresses the needs of the commercial enterprises better than lattice-based MAC [1] and owner-based DAC [2]. Many papers describe ways to model or implement RBAC using the technologies employed by the commercial users. For example, Oracle [3], NetWare [4], Java [5], DG/UX [6], J2EE [7,8], object-oriented systems [9], object-oriented databases [10], MS Windows NT [11], enterprise security management systems [12]. Evidence of RBAC becoming a dominant access control paradigm is the approval of the American National Standard for Information Technology Role-Based Access Control (ANSI RBAC) [13] in 2004.

The ANSI RBAC standard consists of two main parts: the RBAC Reference Model, and the RBAC System and Administrative Functional Specification. The two parts specify four profiles: *Core RBAC* with the minimum set of features included in all RBAC systems, *Hierarchical RBAC* that defines role hierarchies, as well as *Static Separation of Duty Relations* and *Dynamic Separation of Duty Relations* that define static and dynamic constraint relations, accordingly.

At the same time, commercial middleware technologies—such as Common Object Request Broker Architecture (CORBA) [14], COM+ [15], Enterprise Java Beans (EJB) [16]—became mature, with distributed enterprise applications routinely developed with the use of middleware. Each middleware technology, however, comes with its own security subsystem [17–19], sometimes dependent and specific to the underlying operating system (OS). For instance, COM+ security [17] is tied into Microsoft Windows OS and its services.

The ability of 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, such as RBAC, lattice-based MAC [1] and owner-based DAC [2], etc. 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 diversity of distributed applications, the controls seem to be a result of engineering compromises between, among others, perceived customer requirements, the capabilities of the target run-time 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 the documentation of the middleware controls.

[*] Corresponding author.
E-mail address: beznosov@ece.ubc.ca (K. Beznosov).

0920-5489/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2009.12.001

Author's personal copy

Third, the security subsystem semantics in commercial middleware is defined imprecisely, leaving room for misinterpretation. For example, the EJB specification does not address nor dictate how the EJB security roles should be mapped to the operational environment's security principals, leaving the semantics of this mapping up for interpretation by various vendors. Another example is the CORBA OMG specification, where the functionality of various interfaces is not always defined precisely [20]. In this paper, we clarify the semantics of the security subsystem and analyze its ability to support ANSI RBAC for one particular industrial middleware technology—COM+.

We defined the protection state of the access control subsystem of COM+. Our definitions offer precise and unambiguous interpretation of the middleware access control. 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 COM+ supports the family of role-based access control models as defined by ANSI RBAC Standard [13].

We have formalized the authorization-related parts of COM+ [17] into protection state configuration through studying its description and specifications. Then, we used the protection state configuration to analyze the middleware in regards to its support for a particular ANSI RBAC feature, e.g., role hierarchies. When it was possible, we showed how the corresponding ANSI RBAC construct can be expressed in the language of the middleware 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 providers, or enforced 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 in Section 6.

Our analysis shows that COM+ falls short of supporting all ANSI RBAC required functions, although COM+ has better support than CORBA or EJB. The limitations that prevent full support are mainly due to the mismatch between session-oriented nature of RBAC and request-oriented architecture of COM+ and other commercial middleware that we analyzed elsewhere [62,63]. This mismatch calls into question the mandatory support for sessions and related functionality in ANSI RBAC systems. When it comes to multi-host deployments of COM+ systems RBAC administration becomes problematic and the lack of support for enumerating COM+ objects across hosts becomes an impediment. While role hierarchies and separation of duty constraints are not directly supported, they can be, with the help of custom tools for administering COM+ systems.

The work presented in this paper establishes a framework for implementing and assessing implementations of ANSI RBAC using COM+. The results provide directions for COM+ developers supporting ANSI RBAC in their systems and criteria for users and application developers for assessing support for 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 COM+. Section 3 discusses related work. Section 4 formally defines the protection state of the COM+ access control subsystem. A mapping from ANSI RBAC based policies into the COM+ protection state is defined in Section 5. 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 COM+ 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 [21,22]. 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 [23]. A second version was later publicly released in 2001 [24]. 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 [13]. 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. Fig. 1 depicts these RBAC components.

2.1.1.1. 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 [13]:

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: $assigned\_users(r) = \{u \in USRES | (u,r) \in \mathrm{UA}\}$
- $PRMS = 2^{(OPS \times OBS)}$, 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: $assigned\_permissions(r) = \{p \in PRMS | (p,r) \in PA\}$
- $Op(p{:}PRMS) \rightarrow \{op \subseteq OPS\}$, 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: $session\_roles(s_i) \subseteq \{r \in ROLES | (session\_users(s_i), r) \in UA\}$
- $avail\ session\_perms(s{:}SESSIONS) \rightarrow 2^{PRMS}$, the permissions available to a user in a $session = \bigcup_{r \in session_roles(s)} assigned\_permissions(r)$

2.1.1.2. 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.

Author's personal copy

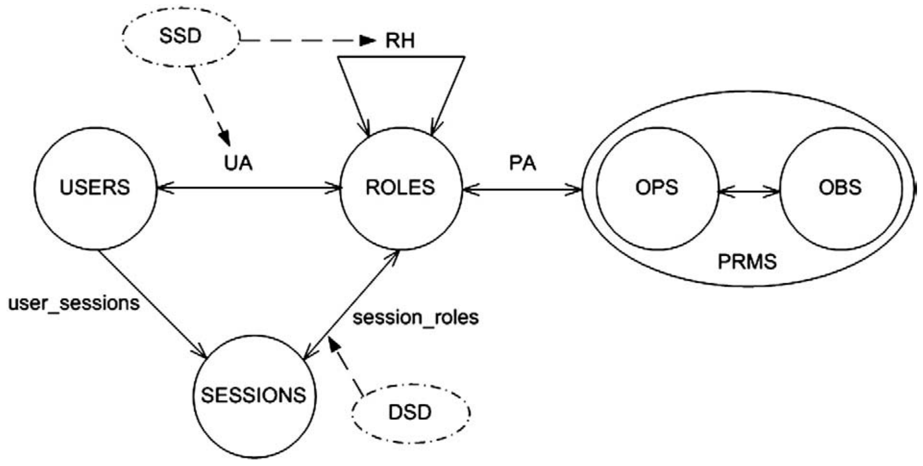


Fig. 1. ANSI RBAC sets, relations, and main functions.

The standard defines two types of role hierarchies. These types are shown in Fig. 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 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:

- $authorized\_users(r) = \{u \in USERS | r' \succeq r, (u, r') \in UA\}$ 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 $r_{\text{senior}} \succeq r_{\text{junior}}$ indicates that r_{senior} inherits all permissions of r_{junior}, and all users of r_{senior} are also users of r_{junior}.

2.1.1.3. 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 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, SetSsd-SetCardinality, and so forth
- DSD Relations administrative operations include CreateDsdSet, AddDsdRoleMember, SetDsd-SetCardinality, 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.

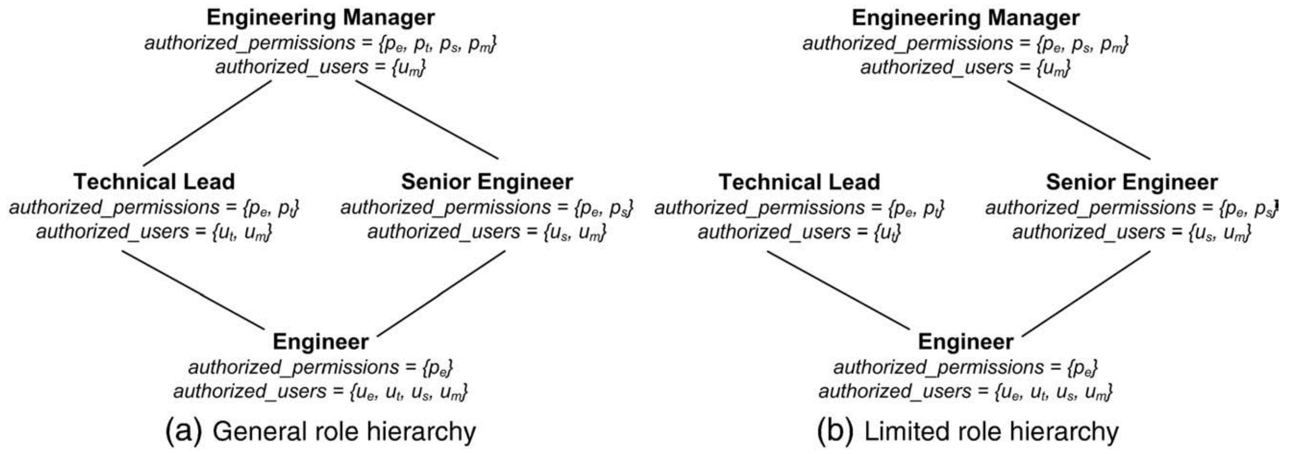

Engineering Manager
authorized_permissions = {pe, pt, ps, pm}
authorized_users = {um}
Technical Lead
authorized_permissions = {pe, pt}
authorized_users = {ut, um}
Senior Engineer
authorized_permissions = {pe, ps}
authorized_users = {us, um}
Engineer
authorized_permissions = {pe}
authorized_users = {ue, ut, us, um}
(a) General role hierarchy
Engineering Manager
authorized_permissions = {pe, ps, pm}
authorized_users = {um}
Technical Lead
authorized_permissions = {pe, pt}
authorized_users = {ut}
Senior Engineer
authorized_permissions = {pe, ps}
authorized_users = {us, um}
Engineer
authorized_permissions = {pe}
authorized_users = {ue, ut, us, um}
(b) Limited role hierarchy

Fig. 2. Examples of Hierarchical RBAC.

Author's personal copy

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 COM+ security

The following sub-section provides a brief informal overview of COM+. More information can be found on the Microsoft MSDN Library web site [25], or various COM+ books, such as [26].

2.2.1. COM+

The Microsoft Component Object Model (COM), Distributed COM (DCOM), and COM+ are all programming frameworks for creating component based software.

The Component Object Model (COM) [27] is an object-based programming model and a binary standard that enables components written in different languages to interoperate. Like CORBA, EJB, and Web Services, COM is based on the principles of information hiding and design by contract [28]. This enables the software components to be reused without any dependencies on the way a component interface is implemented, as long as the implementation satisfies the component's specification. The reuse of these components is based on compiled, binary code. This allows COM components to be upgraded in already deployed systems without having to recompile the applications that use them. Various languages such as C++, Visual Basic and others can be used to develop COM components.

DCOM [29] extends COM with the support for distributed interprocess communication between COM applications; that is, the DCOM architecture enables components or processes to communicate across a computer network. The DCOM communication protocol consists of a set of extensions, layered on the distributed computing environment (DCE) remote procedure call (RPC) specification [30], providing object-oriented RPC (ORPC).

Built on top of COM, and using the DCOM communication protocol, COM+ provides services that handle object and connection pooling, thread synchronization, security, and other resource management tasks. The goal of COM+, like other middleware technologies, is to facilitate application development and deployment without requiring application developers to deal with low-level tasks such as load balancing, distributed transactions, remote method invocation, and so forth. The following are definitions of various COM+ terms.

Interface	defines a set of public operations (a.k.a. methods) that can be invoked by client applications. The interface does not provide any implementation for those methods. In other words, the interface defines a specific way for using the COM+ component. Each interface is identified by a globally unique identifier (GUID).
Class	is a software construct that provides a concrete implementation of one or more interfaces. These classes are compiled into binary files called servers.
Object	is an executable instance of a COM class.
Component	is a software unit of composition with specified interfaces [31]. In COM, a component is compiled code (usually in the form of a library) that complies with the COM standard, and can create COM objects.
Server	is a collection of one or more classes that provide services to clients. These services are provided through the methods of the COM classes. In addition to containing the implementation logic for the classes, servers also support standard COM methods for object activation.
Application	is a group of one or more components. An application can be a server, a client, or a collection of both. There are two types of COM+ applications: server applications and library applications. Server applications run in their own processes whereas library applications run in the same process as their clients.

Given these definitions, we proceed to describe the declarative and the runtime parts of COM+ systems.

2.2.1.1. Declarative part. Since various programming languages are used to develop COM components, a means to describe COM classes and interfaces in a manner independent of a programming language is required. The Microsoft Interface Definition Language (MIDL) [32] is used for this purpose. Once COM+ interfaces are defined in MIDL, an MIDL compiler is used to generate the software code required to implement each interface.

Fig. 3 illustrates the definition of an example interface IEmployee and class CEmployee in MIDL. The first three lines include IDL definitions for the base interface of all COM interfaces, IUnknown; and two custom interfaces used in the example IProject and IExperience. Lines 5–10 define a custom structure for the EmployeeInfo data type. The *BSTR* keyword defines a string. Lines 12–15 define all attributes for the IEmployee interface. The *object* keyword informs the MIDL compiler to generate C++ code to be used to implement COM objects; when this keyword is not used, the MIDL compiler generates code suitable for DCE RPC programs. Since each interface should be uniquely identified, a Universally Unique Identifier (UUID) [30] is used on line 14 to identify this interface.[1]

Lines 16–23 in the example contain the actual definition of the IEmployee interface, which inherits from IUnknown, as required for all COM interfaces. In addition to the methods specified on lines 17–22, the IEmployee interface also inherits the AddRef, Release , and QueryInterface methods defined by IUnknown. The former two methods are used for managing the life cycle of COM+ objects. Since a single class may implement more than one interface, QueryInterface () method is used to obtain a reference to the implementation of a specific interface. For each method parameter, *in* or *out* attributes define parameters to be set by the caller or returned to the caller, respectively. The *HRESULT* is a type that encapsulates a 32-bit return value indicating either successful method execution or a specific error. Lines 27–31 define the class that will be included in the EmployeeLib library and what interfaces the class implements, along with the class attribute(s), such as its *uuid*.

2.2.1.2. Runtime part. COM+ objects have certain attributes that specify their runtime requirements for using various COM+ services, such as synchronization, transactions, security, and so on. These attributes are maintained in a repository referred to as the COM+ catalog. When a client application creates an instance of a COM+ server object, the COM+ catalog is consulted for information required to instantiate the server.

Each COM+ component has a set of attributes that defines the component's run-time needs, such as transactional, threading, and security. A context is a set of runtime constraints associated with one or more COM objects. Each object is associated with only a single context for the duration of the object's life. If the caller and the target object are located in the same context, no constraint checks, including those related to security, are performed; however, if they are running in different contexts, the incoming call goes through an interceptor. The interceptor can do whatever is necessary to satisfy the runtime constraints.

Some of the runtime constraints are related to application's security. COM+ security controls the invocation of object methods in order to allow only authorized users to execute those methods. Several COM+ security features can be used to protect applications. The following section provides an informal description of various COM+ security aspects.

[1] A UUID is equivalent to a GUID. Although the latter is more commonly used in COM+, the keyword uuid is used in IDL files. GUIDs are either created using the guidgen.exe utility, or programmatically using the CoCreateGuid function.

Author's personal copy

```
import "unknwn.idl"
import "project.idl"
import "experience.idl"

typedef struct  EmployeeInfoStruct
{
  BSTR  familyName;
  BSTR*   middleNames;
  BSTR  givenName;
} EmployeeInfo;

[
    object,
    uuid(72d797d5-9f5d-4673-bf7b-ba1955ccb343),
]
interface  IEmployee:IUnknown {
   HRESULT   GetBasicInfo([out] EmployeeInfo* pInfo);
   HRESULT   AssignToProject([in] IProject project);
   HRESULT   UnassignFromProject([in] IProject project);
   HRESULT   AddExperience([in] IExperience experience);
   HRESULT   GetExperience([out] IExperience* pExperience);
   HRESULT   Fire();
}

[ uuid(82f19809-e2c4-4ac3-a7f7-b22da586f906) ]
library  EmployeeLib {
   [ uuid(4a317abe-f759-4b5b-81e1-a34a3a7a927c) ]
   coclass  CEmployee
   {
      interface  IEmployee;
   }
};
```

Fig. 3. An example employee.idl file.

2.2.2. Security subsystem

The security architecture of COM+ employs roles for expressing access control policies. A COM+ role identifies a group of users that share the same permissions to access services provided by a COM+ application. Once roles are defined for an application, the administrator assigns individual Windows user accounts or user groups to roles. Roles are granted permissions to access certain components, interfaces, or methods in the application. Each COM+ application defines its own roles.

COM+ security functions are enforced outside of the application through security interceptors, which are lightweight object proxies. These interceptors ensure that the role attempting to access the server component is authorized to do so. And before clients are authorized to invoke server methods, they may have to be authenticated. COM+ provides various levels of authentication that can be used to secure calls into an application.

Similar to other middleware technologies, a client-side layer (we refer to this layer as the Client Security Service (CSS)) and server side layer (Target Security Service (TSS)) are responsible for enforcing COM+ security policies. The following is a list of various functions provided by these layers.

2.2.2.1. Client Security Service. The CSS is responsible for providing an interface for clients to examine or modify the security settings of a particular connection with an out-of-process COM+ object. For example, the CSS informs the client application what authentication levels are acceptable by the server. The CSS is also responsible for passing client credentials to TSS, when required. On the other hand, if the client application requires the server identity to be authenticated, CSS will enforce this requirement. When a connection is established between the client and the server, CSS cryptographically protects request messages and verifies response messages.

2.2.2.2. Target Security Service. In addition to participating in the authentication protocol negotiation with the client, the TSS supports administration of the server security. TSS can also be given a security descriptor containing a discretionary access control list (DACL) and perform process-wide access checks against the DACL and security tokens of the clients on all incoming calls. TSS interceptor enforces access policies whenever a call is to be dispatched to the application.

2.2.2.3. Implementation of security functions. COM+ provides applications with security features, such as authorization and authentication. In order to secure COM+ applications, authorization and authentication features of COM+ are required at a minimum. COM+ also offers other security features, such as auditing. Based on the security requirements for each application, various COM+ security features can be utilized. The following is a brief description of the minimum requirements to secure a COM+ application. We elaborate more on access control in Section 4.1.

2.2.2.4. Authentication. In COM+, Security Service Providers (SSPs) offer authentication services to both clients and servers. SSPs are implemented as DLLs, and can support a variety of authentication protocols, e.g., Kerberos [33], Windows NT LAN Manager (NTLM) challenge–response authentication protocol [34], public/private key [35] based authentication protocols.

COM+ allows server applications to be configured to require different levels of client authentication. The names of these levels and their descriptions are as follows:

- None: no client authentication is required
- Connect: authentication is required when a connection between the client and server applications is established
- Call: authentication is required on every method invocation
- Packet: authentication is required for each network packet

Author's personal copy

- Packet Integrity: authentication is required for each packet, and data integrity is also checked; and
- Packet Privacy: data encryption, integrity checking, and authentication are required for each packet.

Once the client is authenticated, all COM+ roles to which the client's principal or group are assigned get activated.

2.2.2.5. Administration. The Microsoft Component Services administrative GUI can be used to deploy and configure COM+ applications. The GUI allows administrators to do the following:

- create application specific roles,
- assign users and groups to roles,
- assign permissions to roles,
- specify the minimum level of authentication and message protection a COM+ application would accept,
- enable authorization checks.

The Component Services GUI is built on top of the Component Services Administration Library (COMAdmin) [36]. This means that the functionality provided by the GUI can be also achieved programmatically, allowing administrative tasks to be automated through, for example, scripting.

3. Related work

Over the past decade, there has been no shortage of papers proposing ways to support RBAC. The overwhelming majority of this work, however, is about support for RBAC96 [22], 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 [37], do not provide direct support for RBAC, researchers and developers have been employing either groups (e.g., [38,39]) or user accounts (e.g., [40,41]) to simulate roles. This choice determines whether more than one role can be activated in a session. Role hierarchies are either not supported [37,40] or are simulated by maintaining additional system files with the role hierarchy and various book-keeping data [38,39]. No 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 (i.e., [40,41]): 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 while another role is activated.

We analyzed DB2 [42] and MySQL [43] and updated the analysis of RBAC support in commercial database management systems (DBMS)—conducted by Ramaswamy and Sandhu [44]—with the latest versions of the corresponding systems. Commercial DBMS continue to have the most advanced support for RBAC96. Informix Dynamic Server v7.2 [45], IBM DB2 [42], Sybase Adaptive Server v11.5 [46], and Oracle Enterprise Server v8.0 [47] 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, as in [48–50], or "pulled" by the server from a local or remote database, as in [49,51–54]. 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 [52] has 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) [55] front-end, as in [48,49,51,54], to store role and other information. Role hierarchies are only supported by some implementations, using either manual assignment of permissions of junior roles to senior ones [49], additional files [56], a database [52] or an LDAP server [53,54]. JRBAC-WEB [56] and RBAC/Web [52] also support both types of SoD.

The work most relevant to ours addresses support for RBAC in middleware. Ahn [57] outlines a proposal for enforcing RBAC policies for distributed applications that utilize Microsoft's Distributed Component Object Model (DCOM) [29,58]. His proposal employs the following elements of Windows NT's architecture: (1) 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 in [57], 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, like with the pull model in client–server systems, all assigned roles are activated, leaving no choice to the user. Ahn does not indicate in [57] 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 [59,60] utilizes the PoliCap [61] 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 that contain 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 user's 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 reused on the server for other requests from the same principal. The extent to which RBAC-JaCoWeb conforms to the CORBASec specification is unclear from [59,60]. 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.

In [62,63], we analyze the access control architectures of CORBA and EJB, respectively, and their support for ANSI RBAC. In this paper, we use a similar formalization approach, which employs set theory to abstract the various elements of the access control architecture of

Author's personal copy

each middleware technology. With this abstraction, we then define protection system state and study how ANSI RBAC components can be supported. Our results in [62,63] indicate that the lack of support for ANSI RBAC in both CORBA and EJB is partially attributed to the lack of support for user accounts and their management. However, despite the fact that COM+ offers such support, it still fails to support all Core ANSI RBAC functions. As we show later in this paper, the common limitation of all three middleware technologies that prevents them from supporting the minimal required components of Core RBAC is the lack of support for sessions and role (de)activation. This can be arguably attributed to shortcomings in the ANSI RBAC standard itself.

4. COM+ protection state

In this section, we informally describe access control architecture for COM+. Then, we formally define a configuration of the COM+ protection state. The COM+ concepts presented here are common to both COM+ versions 1.0 [15] and 1.5 [64].

4.1. COM+ access control

Authorizations in COM+ can be specified at the granularity of the component (all class instances), interface, or method. If a client is permitted to access a component as a whole, then that client can invoke any of the component's methods. If the client is permitted to access only certain interfaces in the component, the client will be able to invoke only the methods in those interfaces. The scope of rights on interfaces is limited to the components implementing them, which means that different clients could have different access rights to the same interface implemented by different components. Furthermore, the client can be permitted to invoke only certain methods in an interface.

The built-in security of COM+ provides several features that can be used to protect COM+ applications. COM+ provides two methods of controlling access to resources: *declarative* and *programmatic*. The declarative method can be used to control access to components, interfaces, or even methods. Using the declarative approach, access control can be achieved without having to write code. As various application attributes are stored in the COM+ catalog, administrative tools can be used to manipulate the COM+ catalog and configure access control for various application components. This approach facilitates the decoupling between application logic and security logic.

On the other hand, the programmatic approach can be used to achieve finer granularity of control. Interface methods can be implemented to check role memberships of clients using functions such as IsCallerInRole (). In addition, the following interfaces provide extra information pertaining to security as follows:

ISecurityCallContext provides access to information on the current method invocation.

ISecurityCallersColl provides access to information about individual callers in the collection of callers.

ISecurityIdentityColl provides access to the collection of information pertaining to the caller's identity.

The TSS controls client access to COM+ server applications. Based on the server application's access policy, security checks are performed before a client's call is successfully dispatched to a server object. For example, if the COM+ server is not running, the client needs to have sufficient permissions to launch the server application before any method can be invoked. TSS checks client permissions to activate the server process. In addition, when a call enters the running server process, further access checks are performed.

Access permissions are enforced using roles. The Component Services GUI allows administrators to create roles for a specific application when deploying it, and to map users to those roles. Once the roles are created, the administrator can choose which components, interfaces and methods in the COM+ application can be accessed by the users assigned to those roles. Fig. 4 uses the Unified Modeling Language (UML) notation to summarize the relationships among authorization-related elements of the COM+ access control architecture, where *account* and *group* in the figure refer to Microsoft Windows based user account and users group, respectively.

An example of role-permission assignments in a COM+ application is shown in Table 1. The first row illustrates assignments of permissions to invoke method m_1 on interface i_1 in component c_1 to roles r_1 and r_2. This indicates that only principals with roles r_1 and/or r_2 are allowed to invoke method m_1 on c_1. The second row in the table illustrates an assignment of permission to invoke all methods on interfaces i_1 and i_2 in component c_2 to role r_3. The last row shows an example of allowing roles r_1, r_2, and r_3 to invoke all methods provided by component c_3.

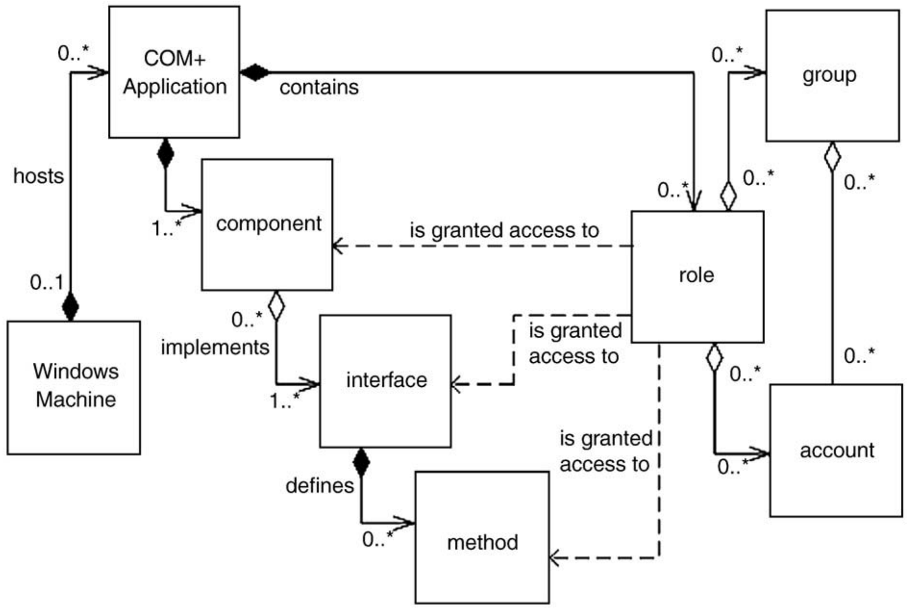


Fig. 4. UML model of COM+ access control architecture.

Author's personal copy

Table 1
Examples of role-permission assignment in a COM+ application.

Roles	Methods
r_1, r_2	$c_1i_1m_1$
r_3	c_2i_1*, c_2i_2*
r_1, r_2, r_3	c_3*

4.2. Formalization of the protection state

In this section, we formalize the protection state of a COM+ system. In this formalization, we attempt to preserve the COM+ terminology.

Definition 2 [COM+ Protection State]. A configuration of a COM+ system protection state for a given application is a tuple (R,U,G,UGA,C,I,M,UA,GA,PA,*isSecurityEnabled*,*user_roles*) interpreted as follows:

- R is a set of the COM+ security roles as defined in the COM+ catalog for a specific application.
- U is a set of users.
- G is a set of user groups.
- $UGA \subseteq U \times G$ is a many-to-many relation of users to groups.
- C is a set of COM+ components for a specific application.
- I is a set of COM+ interfaces provided by the COM+ components in a specific application.
- M' is a set of COM+ method signatures, $\{m_1, m_2, \ldots\}$
- $M \subseteq \mathrm{C} \times \mathrm{I} \times \mathrm{M}'$ is a set of COM+ methods implemented for the provided interfaces. Members of this set are denoted $c_j.i_k.m_l$, where $c_j \in C$, $i_k \in I$, and $m_l \in M$. The set also includes the elements $c_j.i_k.m_*$, which are all methods in interface i_k provided by component c_j; and the elements $c_j.i_*.m_*$, which are all methods in all interfaces provided by component c_j.
- $UA \subseteq R \times U$ is a relation of COM+ security roles to users.
- $GA \subseteq R \times G$ is a relation of COM+ security roles to groups.
- $PA \subseteq R \times M$ is a role-to-method relation.
- *isSecurityEnabled* is a boolean indicating whether access control should be enforced.
- $direct\_user\_roles(u{:}U){:}U \rightarrow 2^R$ is a function mapping each user u to a set of roles that u is directly assigned to. Formally, $direct\_user\_roles(u{:}U) \subseteq \{r|(r,u) \in UA\}$.
- $group\_roles(g{:}G){:}G \rightarrow 2^R$ is a function mapping each group g to a set of roles that g is directly assigned to. Formally, $group\_roles(g{:}G) \subseteq \{r|(r,g) \in GA\}$.
- $user\_groups(u{:}U){:}U \rightarrow 2^G$ is a function mapping each user u to a set of groups that u is a member of. Formally, $user\_groups(u{:}U) \subseteq \{g|(u,g) \in UGA\}$.
- $indirect\_user\_roles(u{:}U)$ is a function mapping user groups to a set of roles. Formally, $indirect\_user\_roles(u{:}U) \subseteq \cup_{g \in user\_groups(u)} \{r|(r,g) \in GA\}$, where these roles are indirectly assigned to the user because the roles are (directly) assigned to the groups to which the user belongs.
- $user\_roles(u{:}U) \equiv direct\_user\_roles(u) \cup indirect\_user\_roles(u)$ is a set of all user roles.

Given the protection state of a COM+ application, Algorithm 1 defines the outcome of an access control decision. Line 1 defines the signature of the Authorize function, which takes in an element from the power set of all available roles, and an element from the set of COM+ methods. The Authorize function returns either allow or deny. If isSecurityEnabled is true, it means that the application deployer or administrator explicitly enabled component level access checks. In such case, the algorithm proceeds to check the calling user's role membership, as shown on lines 5–9. Line 6 shows that if any of the roles the user is assigned to has explicit permission to invoke the method m_1 in interface i_k in component c_j, the algorithm will authorize the user to invoke the method in question. Furthermore, if any of the roles the user is assigned to has implicit permission to invoke the method in question, the algorithm will also authorize the user to invoke the method. By implicit permission we mean a permission that is inferred from either allowing the role to invoke all methods in the specific interface i_k that m_1 belongs to, or allowing the role to invoke all methods in all interfaces in a specific component c_j that m_1 is part of. If none of these conditions (either explicit or implicit permissions) is true, the authorization algorithm denies the user its request to invoke the method, as shown on line 10.

Algorithm 1. Authorization decision in COM+

```
 1: Authorize($p : 2^R$,$c.i.m : M$) → {allow,deny}
 2: if $isSecurityEnabled \neq true$ then
 3: return allow
 4: else
 5: for all $r \in p$ do
 6: if $(r,c.i.m) \in PA \vee (r,c.i.m_*) \in PA \vee (r,c.i_*.m_*) \in PA$ then
 7: return allow
 8: end if
 9: end for
10: return deny
11: end if
```

5. Support for ANSI RBAC in COM+

For a system to comply with ANSI RBAC, Core RBAC must be implemented at a minimum; the other three RBAC components (Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty), as described in Section 2.1, are optional. In Section 5.1, we first examine the extent to which a COM+ protection state—as formalized in Definition 2—can support each of the four ANSI RBAC components. In Section 5.3, we illustrate our formalization with an example. In Section 5.4, we then analyze the degree to which COM+ supports the functional specification of ANSI RBAC.

5.1. Reference model components

5.1.1. Core RBAC

Defined in Section 4, the COM+ protection state configuration can realize security policies that are based on Core RBAC as follows. Core RBAC *ROLES* set maps directly to COM+ security roles (set R). In COM+, roles can be assigned to individual user accounts as well as Windows groups. In RBAC, however, there is no concept of groups. To map the COM+ use of groups into ANSI RBAC, we introduced function $users(g{:}G)$, which returns all users in a given group. Given this function, the *USERS* set in RBAC would map to the union of U and all sets that $users(g{:}G)$ function returns for all groups. RBAC permission assignment (PA) is equivalent to the one in COM+. More formally, we define Core RBAC in the language of the COM+ protection system as follows:

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

- $users(g{:}G) \rightarrow \{u \in U\}$, a function that returns a set of users in a certain group.
- $USERS = U \cup \bigcup_{g \in G} users(g : G)$ is a set of individual users and all group users, where members of USERS are user accounts in Windows and users in all groups.
- $ROLES = R$ is a set of roles, where members of $ROLES$ are the roles defined for a specific COM+ application.

Author's personal copy

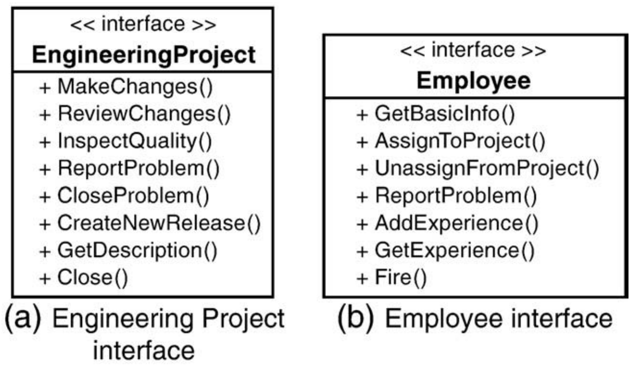


Fig. 5. Example COM+ interfaces.

- *OPS* is a set of operations, where members of this set are operations that can be invoked on COM+ components; for example, $OPS = \{m_x, m_y, m_*, \ldots\}$.
- $OBS \subseteq C \times I$ is a set of objects, where these objects are defined to be specific interfaces on certain components, or all interfaces on a certain component; for example, $OBS = \{c_1 i_y, c_1 i_z, c_1 i_*, c_2 i_x, c_2 i_*, \ldots\}$.
- $UA = USERS \times ROLES$, is a many-to-many assignment of users to roles.
- $assigned\_users(r{:}ROLES) = \{u \in USERS | (u,r) \in UA\}$ is a function that returns the set of users in *USERS* that are assigned to the given role *r*.
- $PRMS \subseteq OPS \times OBS$ is a set of permissions to invoke COM+ interface methods for certain components. The existence of $c_j i_k m_l$, $c_j i_k m_*$ or $c_j i_* m_*$ in *PRMS* grants permission to invoke a specific method m_1, all methods in interface i_k or all methods in all interfaces in component c_j, respectively; for example, $PRMS = \{c_1 i_y m_x, c_1 i_z m_*, c_2 i_x m_y, \ldots\}$.
- $PA \subseteq PRMS \times ROLES$, a many-to-many assignment of permissions to COM+ roles.
- $assigned\_permissions(r{:}ROLES) = \{p \in PRMS | (p,r) \in PA\}$ is a function that returns the set of permissions in *PRMS* that are assigned to the given role *r*.
- $Op(p{:}PERMS) \rightarrow \{op \in OPS\}$ is a function that returns a set of operations that are associated with the given permission *p*, e.g., $Op(c_j i_k m_l) = m_l$.
- $Ob(p{:}PERMS) \rightarrow \{ob \in OBS\}$ is a function that returns a set of objects that are associated with the given permission *p*, e.g., $Ob(c_j i_k m_l) = \{c_j i_k\}$.
- *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 COM+ application. Like with many other systems, in a COM+ application environment, all assigned roles are activated for a user once the user is authenticated.
- $session\_users(s{:}SESSIONS) \rightarrow USERS$, 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 permissions available to a user in a $session = \bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$.

As shown in Definition 3, most of the elements required to support Core RBAC are already provided in the COM+ protection state (Definition 2). However, the mapping between RBAC elements and the COM+ protection state is not straightforward.

For example, user groups do not exist in RBAC. Expanding groups into sets of users is required in order to have group support in RBAC. Another option would be to use COM+ groups to define the RBAC USERS set as $U \times G$; however, because there is a user-to-group relationship in COM+, this mapping would not be ideal.

Another example is the *OPS* set. This RBAC set does not directly map to COM+ operations. COM+ operations are uniquely defined by an interface and a method. However, defining RBAC *OPS* as a subset of $I \times M$ would not result in the correct mapping of RBAC *PRMS*. The mapping between the *OBS* and COM+ protection state elements is not straightforward, either. Objects in COM+ refer to instances of classes; however, in the RBAC terminology, an object is the resource being protected. In COM+ what is being protected is either a component, interface, or method. A straight forward mapping would define RBAC *OBS* in terms of $C \times I \times M$; however, this mapping would be incorrect, as it would define *PRMS* as $OPS \times C \times I \times M$.

Finally, the elements related to *SESSIONS* are not addressed in Definition 2. This is due to the fact that user sessions are specific to the MS Windows platform and are not specific to COM+. Since all operating system processes on an MS Windows platform must be associated with a logon session, these sessions are handled by the operating system and are transparent to the COM+ application.

5.1.2. Hierarchical RBAC

General Role Hierarchies and Limited Role Hierarchies comprise the Hierarchical RBAC component of the ANSI RBAC Reference Model. Both role hierarchies are formally defined in terms of the sets and relations of the Core RBAC component. These components are

1. Anyone in the organization can look up an employee's name.
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 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 engineering projects.
8. Everyone who is an administrator can get the description of a project, and look up the name and experience of any employee.

Fig. 6. Sample authorization policy for the example COM+ application describing what actions are allowed. All other actions are denied.

Author's personal copy

described in Section 2.1.1. Neither the COM+ catalog nor the administrative tool that is part of the COM+ environment directly support creating hierarchical relationships between roles.

5.1.3. Constrained RBAC

Static Separation of Duty (SSD) and Dynamic Separation of Duty (DSD) relations are part of the Constrained RBAC component of ANSI RBAC [13]. Similar to the Hierarchical RBAC component, these relations are defined in terms of Core RBAC elements. Essentially, 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*). The COM+ catalog does not allow for specifying any constraints on user-to-role assignments, whether static or dynamic; neither does it allow for specifying any constraints on role activation. As such, SSD and DSD are not supported in COM+.

5.2. Translating RBAC Policies to COM+

In Definition 2 and Definition 3 we presented a protection state for COM+ systems, and how Core RBAC can be modeled in the language of the COM+ protection state, respectively. In this section, we present an algorithm that translates an arbitrary RBAC policy into a COM+ protection state.

Algorithm 2 formalizes the translation from an RBAC policy to the COM+ protection state defined in Definition 2. For clarity, we identify the RBAC sets in the algorithm with an RBAC subscript. The algorithm requires the following two functions defined as follows:

- *component*(*o:OBS*) → *c:* returns the component corresponding to a given object *o*.
- *interface*(*o:OBS*) → *i:* returns the interface corresponding to a given object *o*.

5.3. Example

In this section we present an example that illustrates how ANSI RBAC can be supported in a COM+ system as discussed earlier. This example is a simple COM+ application that maintains employee and engineering project records in an engineering company. The application allows users to perform various operations on the project and employee records, based on the users' roles in the company. The application consists of a single component, EngineeringProjectService (EPS), which supports the following COM+ interfaces: Engineering-Project, and Employee. These interfaces are shown in Fig. 5. In this example, we define seven different user roles. Based on these roles and according to the policies listed in Fig. 6, users are allowed to invoke various methods in this application. 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* overseas and leads the development of a project.
- *Director* is an engineering department director.
- *Administrator* represents all employees who belong to upper management as well as operations.

Algorithm 2. Operational definition of translating from an ANSI RBAC system state to the one of COM+.

```
1:  {Initialize COM+ sets and relations.}
2:  $R \leftarrow ROLES_{RBAC}$
3:  $U \leftarrow USERS_{RBAC}$
4:  $G \leftarrow \emptyset$
5:  $UGA \leftarrow U$
6:  $C \leftarrow \emptyset$
7:  $I \leftarrow \emptyset$
8:  $M \leftarrow \emptyset$
9:  $UA \leftarrow \emptyset$
10: $GA \leftarrow \emptyset$
11: $PA \leftarrow \emptyset$
12: $isSecurityEnabled \leftarrow$ true
13: for all $p \in PRMS_{RBAC}$ do
14:   for all $(opr, obj) \in p$ do
15:     $c \leftarrow component(obj)$
16:     $i \leftarrow interface(obj)$
17:     $C \leftarrow C \cup \{c\}$
18:     $I \leftarrow I \cup \{i\}$
19:     $M \leftarrow M \cup \{opr\}$
20:   end for
21: end for
22: for all $pa \in PA_{RBAC}$ do
23:   for all $((opr; obj); r) \in pa$ do
```

Table 2
Example COM+ role–method permissions.

COM+ roles	COM+ methods													
	EngineeringProject								Employee					
	MakeChanges()	ReviewChanges()	InspectQuality()	ReportProblem()	CloseProblem()	CreateNewRelease()	GetDescription()	Close()	get_basic_info()	AssignToProject()	UnassignFromProject()	AddExperience()	GetExperience()	Fire()
Employee									✓				✓	
Engineering Department				✓			✓		✓				✓	
Engineer	✓	✓							✓				✓	
Product Engineer						✓			✓				✓	
Quality Engineer			✓						✓				✓	
Project Lead					✓				✓				✓	
Director								✓	✓	✓	✓	✓	✓	✓
Administrator							✓		✓				✓	

Author's personal copy

```
24: $UA \leftarrow UA \cup \{r\} \times assigned\_users(r)$
25: $PA \leftarrow PA \cup \{(r; opr)\}$
26: end for
27: end for
```

System access control policy that defines what actions each role is allowed to perform is summarized in Table 2, where a check mark ("✓") denotes a granted permission for a specific COM+ role to execute the corresponding method. Tables in Fig. 7 show examples of users-to-roles assignments, groups-to-roles assignments, and an example of system users, and their group memberships from the underlying MS Windows operating system environment. The following is a formalization of this example system's protection state as in Definition 2.

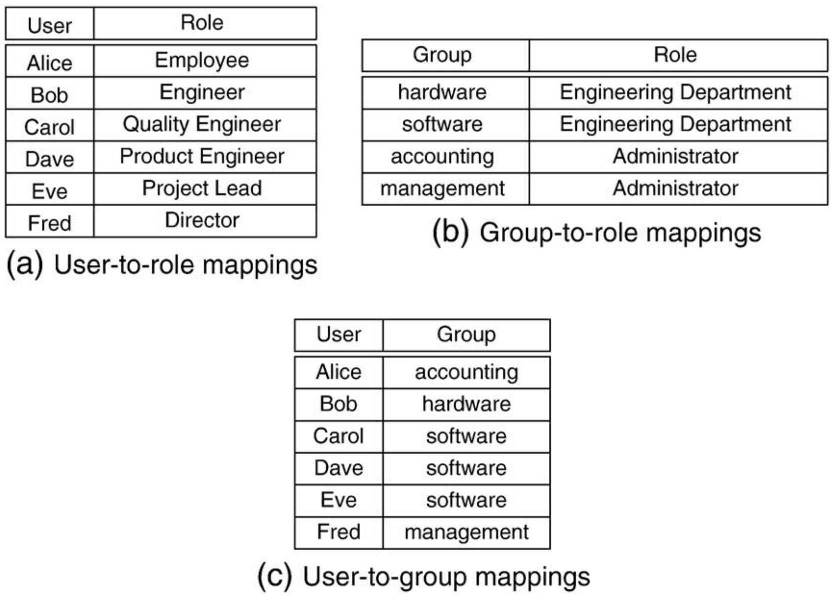
- R = {Employee, Engineering Department, Engineer, Product Engineer, Quality Engineer, Project Lead, Director, Administrator}
- U = {Alice, Bob, Carol, Dave, Eve, Fred}
- G = {hardware, software, accounting, management}
- UGA = {(Alice, accounting), (Bob, hardware), (Carol, software), (Dave, software), (Eve, software), (Fred, management)}
- C = {EPS}
- I = {EngineeringProject, Employee}
- M = {EPS.EngineeringProject.MakeChanges, EPS.EngineeringProject.ReviewChanges, EPS.EngineeringProject.InspectQuality, EPS.EngineeringProject.ReportProblem, EPS.EngineeringProjcet.CloseProblem, EPS.EngineeringProject.CreateNewRelease, EPS.EngineeringProject.GetDescription, EPS.EngineeringProject.Close, EPS.Employee.GetBasicInfo, EPS.Employee.AssignToProject, EPS.Employee.UnassignFromProject, EPS.Employee.AddExperience, EPS.Employee.GetExperience, EPS.Employee.Fire }
- UA = {(Employee, Alice), (Engineer, Rob), (Quality Engineer, Carol), (Product Engineer, Dave), (Project Lead, Eve), (Director, Fred)}
- GA = {(Engineering Department, hardware), (Engineering Department, hardware), (Administrator, accounting), (Administrator, management)}
- PA = {(Employee, EPS.Employee.GetBasicInfo), (Employee, EPS.Employee.GetExperience), (Engineering Department, EPS.EngineeringProject.ReportProblem), (Engineering Department, EPS.Employee.GetBasicInfo), (Engineering Department, EPS.Employee.GetExperience) (Engineer, EPS.EngineeringProject.MakeChanges), (Engineer, EPS.EngineeringProject.ReviewChanges), (Engineer, EPS.Employee.GetBasicInfo), (Engineer, EPS.Employee.GetExperience), (Product Engineer, EPS.EngineeringProject.CreateNewRelease), (Product Engineer, EPS.Employee.GetBasicInfo), (Product Engineer, EPS.Employee.GetExperience), (Quality Engineer, EPS.EngineeringProject.InspectQuality), (Quality Engineer, EPS.Employee.GetBasicInfo), (Quality Engineer, EPS.Employee.GetExperience), (Project Lead, EPS.EngineeringProject.CloseProblem), (Project Lead, EPS.Employee.GetBasicInfo), (Project Lead, EPS.Employee.GetExperience), (Director, EPS.EngineeringProject.Close), (Director, EPS.Employee.*), (Administrator, EPS.EngineeringProject.GetDescription), (Administrator, EPS.Employee.GetBasicInfo), (Administrator, EPS.Employee.GetExperience)}
- $isSecurityEnabled$ = true

In this example, the $user\_roles(u{:}U)$ function, as formalized in Definition 2, returns the roles assigned to a specific user whether this assignment is direct as specified in UA, or by inference using the information from the user's group assignment as specified in UGA and the user's group's role assignment as specified in GA. For example, $user\_roles(Fred) = \{Director, Administrator\}$, where the $(Director, Fred) \in UA$, and $(Fred, management) \in UGA$ and $(Administrator, management) \in GA$. In accordance with Definition 3, we also identify the following sets in order to support Core RBAC.

- $USERS$ = {Alice, Bob, Carol, Dave, Eve, Fred} ∪ {Bob} ∪ {Carol, Dave, Eve} ∪ {Alice} ∪ {Fred} = {Alice, Bob, Carol, Dave, Eve, Fred}
- OPS = {MakeChanges, ReviewChanges, InspectQuality, ReportProblem, CloseProblem, CreateNewRelease, GetDescription, Close, GetBasicInfo, AssignToProject, UnassignFromProject, AddExperience, GetExperience, Employee.Fire }
- $PRMS = M$

5.4. Functional specification

In this section, we examine the ability of COM+ middleware to support ANSI RBAC administrative operations for the creation and maintenance of RBAC sets and relations, review functions for



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

(a) User-to-role mappings

Group	Role
hardware	Engineering Department
software	Engineering Department
accounting	Administrator
management	Administrator

(b) Group-to-role mappings

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

(c) User-to-group mappings

Fig. 7. Example COM+ application user, group, and role mappings.

Author's personal copy

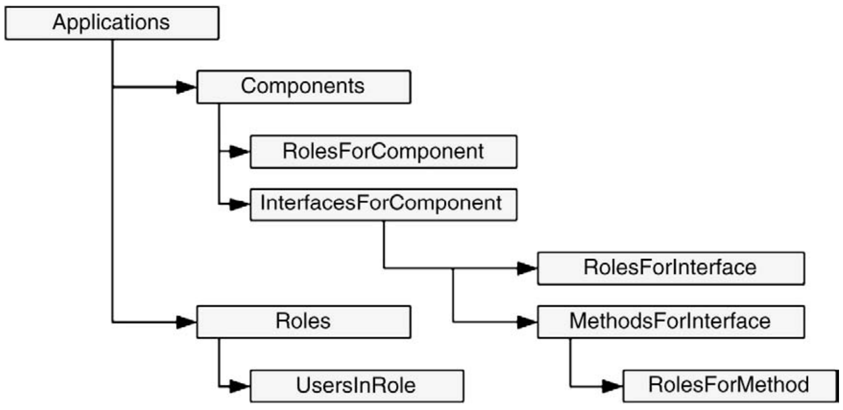


Fig. 8. COM+ administration collections.

performing administrative queries, and system functions for creating and managing user sessions and making access control decisions. For those RBAC functions that are supported by COM+, we provide example pseudo-code fragments (with syntax very similar to C++) showing how these functions can be implemented. In these fragments, we use procedures from COM+ Component Services Administration Library (COMAdmin) [36] and Network Management API [65].

COMAdmin provides a variety of classes and programming interfaces for managing COM+ applications, as well as for manipulating various attributes stored in the COM+ catalog. COMAdminCatalog is one of the classes used to access COM+ configuration data stored in the COM+ catalog. The class implements two interfaces: ICOMAdminCatalog and ICOMAdminCatalog2; the latter is available only in COM+ version 1.5. The COMAdminCatalog provides the GetCollection method, which can be used to retrieve COMAdminCatalogCollection objects that represent COM+ applications, COM+ components, and so on. Each COMAdminCatalogCollection can be further queried for more information or "sub-collections," or manipulated as required. Fig. 8, illustrates the relationships among various collections that are referenced throughout this section in the pseudo-code fragments. The arrows indicate the ability to navigate from one collection to another using the GetCollection method of the COMAdminCatalogCollection object. In addition to the GetCollection method, the COMAdminCatalogCollection class provides Add and Remove methods for adding and removing objects from a certain collection. For example, to add a new COM+ role to a certain application, an algorithm similar to the one outlined in Fig. 9 can be used. Due to space limitations, we omit definitions of simple custom helper procedures such as GetCatalogObject, FindApplication, and CreateRoleObject. For the same reason, we also omit invocations of Populate method on the COM+ administration collections.

5.4.1. Administrative commands for Core RBAC

AddUser, DeleteUser create a new user and delete an existing user from the system. These commands can be implemented using the NetUserAdd, and the NetUserDel methods of the Win32 Network Management APIs. The NetUserAdd method adds a new user account to a system given the system's Domain Name Service (DNS) [66] name, or its NetBIOS [67] name. On the other hand, the NetUserDel

```
AddRole(in role, in application)
{
  COMAdminCatalog comAdminObj;
  COMAdminCatalogCollection appsColl, rolesColl;
  COMAdminCatalogObject roleObj, appObj;

  /* get the com admin object */
  comAdminObj = GetCatalogObject();

  /* get a reference to the applications collection */
  appsColl = comAdminObj.GetCollection(''Applications'');

  /* find the specific application in the collection */
  appObj = FindApplication(appsColl, application);

  /* get the roles collection for that application */
  rolesColl = appsColl.GetCollection(''Roles'', appObj.Key);

  /* create a role object and add it to
   * the Roles collection for that application */
  roleObj = CreateRoleObject(role);
  rolesColl.Add(roleObj);
}
```

Fig. 9. Pseudo-code for adding a COM+ role to an application.

Author's personal copy

```
DeleteRole(in role, in application)
{
  int roleObjIndex;

  /* same code as lines 3–17 of AddRole */
  ...

  /* find the index of the role object to be deleted */
  roleObjIndex = FindRoleIndex(rolesColl, role);
  rolesColl.Remove(roleObjIndex);
}
```

Fig. 10. Pseudo-code for removing a COM+ role from an application.

method deletes a user account from the system. Considering Definition 2, the NetUserAdd and NetDelUser methods can be used to manipulate U by adding or deleting elements. In addition to these methods, the Win32 Network Management procedures NetGroupAdd, NetGroupAddUser, NetGroupDel, and NetGroupDelUser can be used to manipulate *G* and *UGA*.

AddRole, DeleteRole methods enable creation and deletion of roles. These methods can be implemented by manipulating the COM+ catalog using the Add and Remove methods on the COM+ Roles collection. As shown in Fig. 8, this collection objects can be accessed by first getting a reference to the Applications collection, then obtaining a reference to the Roles collection. Figs. 9 and 10 provide pseudo-code for the AddRole and DeleteRole functions, respectively.

AssignUser, DeassignUser allow user-to-role assignments to be created and deleted. These commands can be implemented by manipulating the UsersInRole COM+ collection, as shown in Fig. 8. Once the reference to this collection is obtained, the Add or Remove methods can be used to assign a user to a role, or to remove a user from a role assignment. Figs. 11 and 12 provide pseudo-code for implementing AssignUser and DeassignUser functions, respectively.

GrantPermission, RevokePermission are used to grant or revoke the permission to invoke an operation on an object to a role. These methods can be implemented by manipulating the RolesForComponent, RolesForInterface, and RolesForMethod COM+ collections. Fig. 8 shows how to navigate from the Applications collection to one of these role related collections. Using the Add and Remove methods provided by each one of these collections, a role object can be added to or removed from the collection. By adding a role to the RolesForComponent collection, for example, the role is granted permission to invoke all of the COM+ component's methods. On the other hand, adding a role to the RolesForMethod collection allows the role to invoke only a specific method (assuming the role is not added to the RolesForComponent or RolesForInterface collections). Figs. 13 and 14 show pseudo-code for implementing GrantPermission and RevokePermission functions, respectively.

CreateSession, DeleteSession, AddActiveRole, DropActiveRole are used to create and delete a session for a user, and activate or deactivate a role for a user in a given session, respectively. Sessions are handled by Windows, and are created upon user authentication. Role activation and deactivation are also handled by Windows, and are transparent to the user application. However, once the session is established, roles cannot be (de)activated for that session. We could not find Windows or COM+ APIs that would provide support for implementing these functions.

CheckAccess indicates whether a user is allowed or is not allowed to perform a given operation on a given object. Algorithm 1 defined in Section 4.2 can be used to implement CheckAccess. However, since user sessions are handled transparently by Windows, the Authorize function in Algorithm 1 doesn't take a user session as an input parameter.

AssignedUsers, AssignedRoles return the set of users assigned to a specific role, and the set of roles assigned to a specific user, respectively. These functions can be implemented by querying various COM+ collections for their items.

```
AssignUser(in user, in role, in application)
{
  COMAdminCatalogCollection usersColl;
  COMAdminCatalogObject userObj;

  /* same code as lines 3–17 of AddRole */
  ...

  /* find the role in the collection */
  roleObj = FindRole(rolesColl, role);

  /* get the users collection for this role */
  usersColl = rolesColl.GetCollection(''UsersInRole'', roleObj.Key);

  /* create a user object and add it to
   * the UsersInRole collection for the specified role */
  userObj = CreateUserObject(user);
  usersColl.Add(userObj);
}
```

Fig. 11. Pseudo-code for assigning a user to a specific role for a COM+ application.

Author's personal copy

```
DeassignUser(in user, in role, in application)
{
  int userObjIndex;

  /* same code as lines 3–13 for AssignUser */
  ...

  /* find the index of the role object to be deleted */
  userObjIndex = FindUserIndex(usersColl, user);
  usersColl.Remove(userObjIndex);
}
```

Fig. 12. Pseudo-code for deleting the assignment of a user to a role for a COM+ application.

```
GrantPermission(in component, in interface, in method, in role, in application)
{
  COMAdminCatalog comAdminObj;
  COMAdminCatalogCollection appsColl, componentsColl, interfacesColl, methodsColl;
  COMAdminCatalogObject appObj, componentObj, interfaceObj, methodObj;

  /* get the com admin object */
  comAdminObj = GetCatalogObject();

  /* get a reference to the applications collection */
  appsColl = comAdminObj.GetCollection(``Applications'');

  /* find the specific application in the collection */
  appObj = FindApplication(appsColl, application);

  /* get the components collection for that application */
  componentsColl = appsColl.GetCollection(``Components'', appObj.Key);

  componentObj = FindComponent(componentsColl, component);

  /* get the interfaces collection for the component */
  interfacesColl = componentsColl.GetCollection(``InterfacesForComponent'',
                                                 componentObj.Key);

  interfaceObj = FindInterface(interfacesColl, interface);

  /* get the methods collection for the interface */
  methodsColl = interfacesColl.GetCollection(``MethodsForInterface'',
                                              interfaceObj.Key);

  methodObj = FindMethod(methodsColl, method);

  /* get the roles collection for the method */
  rolesColl = methodsColl.GetCollection(``RolesForMethod'', methodObj.Key);

  /* create a role object and add it to
   * the collection for that method */
  roleObj = CreateRoleObject(role);
  rolesColl.Add(roleObj);
}
```

Fig. 13. Pseudo-code for granting permission to a COM+ role to perform a specific operation.

```
RevokePermission(in component, in interface, in method, in role, in application)
{
  int roleIndex;

  /* same code as lines 3–34 for GrantPermission */
   ...

  /* find the index of the role object to be deleted */
  roleObjIndex = FindRoleIndex(rolesColl, role);
  rolesColl.Remove(roleObjIndex);
}
```

Fig. 14. Pseudo-code for revoking permission from a COM+ role to perform a specific operation.

Author's personal copy

```
AssignedUsers( in role, in application, out users )
{
  int index;

  /* same code as lines 3-13 for AssignUser */
  ...

  /* add each item in the users collection to the output set */
  for (index=0; index < usersColl.Count; index++)
     users[index] = usersColl.Item(index);
}
```

Fig. 15. Pseudo-code for returning the set of users assigned to a given role for a COM+ application.

AssignedUsers can be implemented directly by querying the UsersInRole collection; this is shown in Fig. 15. AssignedRoles can be implemented by searching for all roles assigned to a given user, also using the UsersInRole collection; this is shown in Fig. 16.

5.5. Advanced review functions for Core RBAC

RolePermissions, UserPermissions return the set of permissions granted to a given role and user, respectively. RolePermissions can be implemented by identifying which roles are assigned to which component, interface, or method. This is done by querying the RolesForComponent, RolesForInterface, and RolesForMethod collections (see Fig. 8). UserPermissions, on the other hand, can be implemented by identifying which users are assigned to which roles using the UsersInRole collection; then RolePermissions can be used for each one of the user's roles to determine all permissions assigned to the user by knowing which roles the user is assigned to. Querying the appropriate collections to retrieve permissions can be programmed using similar steps as the ones used in Fig. 13.

SessionRoles, SessionPermissions return the active roles and permissions associated with a session. Since all roles assigned to the user are activated when the user's session is created, the AssignedRoles and UserPermissions methods discussed previously can be used to implement SessionRoles and SessionPermissions, respectively.

RoleOperationsOnObject, UserOperationsOnObject return the operations that a given role or user can perform on an object. These functions can be implemented by querying the RolesForComponent, RolesForInterface , and RolesForMethod collections to identify which methods the given role is allowed to invoke. Similarly, the UserOperationsOnObject can be implemented by first identifying which roles are assigned to the given user using the UsersInRole collection, then identifying which methods each one of the user's roles is allowed to invoke. Querying the appropriate collections to retrieve permissions can be programmed using similar steps as the ones used in Fig. 13.

Table 3 provides a summary of the above discussion. Support for ANSI Core RBAC functions is classified in two categories as follows: the first category identifies the functions that can be supported using APIs built into the COM+ operating environment; the second category contains the functions that are not supported. Even if the operating environment provides APIs that are capable of implementing functionality to support sessions and role activation, such an implementation would be proprietary and completely outside the scope of the COM+ standard. As such, the ANSI Core RBAC functions in the last column of the table are flagged as *unsupported*.

6. Discussion

In addition to our analysis of ANSI RBAC support in COM+, we also analyzed support for ANSI RBAC in two other commercial middleware technologies CORBA [14] and EJB [16]. While details of our analysis for these middleware can be found elsewhere [62,63], a summary for all three is provided in Table 4. Results of our investigation for COM+

```
AssignedRoles( in user, in application, out roles )
{
  int rolesIndex, usersIndex;
  int setIndex=0;

  /* same code as lines 3-13 for AssignUser */
  ...

  /* loop through all roles; if for a specific role the given user is assigned, add
     the role to the output set */
  for (rolesIndex=0; rolesIndex < rolesColl.Count; rolesIndex++)
     for (usersIndex=0; usersIndex < usersColl.Count; usersIndex++)
        if ( user == usersColl.Item(usersIndex) )
        {
           roles[setIndex++] = rolesColl.Item(rolesIndex);
           break;
        }
}
```

Fig. 16. Pseudo-code for returning the set of roles assigned to a given user for a COM+ application.

Author's personal copy

Table 3
Functions defined by ANSI Core RBAC and their support in COM+.

Core RBAC functions	Built-in API support		Supported by Win32 Network Management APIs
	Supported by COMAdmin library	Supported by Win32 Network Management APIs	
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	✓		

Table 4
ANSI Core RBAC functions and their support in various middleware technologies.

Core RBAC functions	CORBA	EJB	COM+
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		✓	

suggest that it also falls short of fully supporting ANSI RBAC. Among the three technologies, however, it becomes the closest, providing support for about 80% of ANSI Core RBAC functions.

The biggest missing group is Supporting System Functions, which are required for session management [13, p.37]. Specifically, the missing functions from this group are responsible for session creation/deletion and for role activation/deactivation. The last two are intended to be used by users for altering the set of active roles in the session.

It was interesting to find out that none of the analyzed middleware technologies (see Table 4) support either of these four functions. The only explanation we found is that ANSI RBAC (as well as RBAC96) is session-oriented, whereas CORBA, EJB, and COM+ are request-oriented. In RBAC, a session is used as a holder for the activated roles, which can be retrieved using *session_roles* function. Consequently, access requests are tied to a particular session via *CheckAccess*, whose one of the input parameters is *session*. Role activation and deactivation are also tied into the notion of a session. As such, these two functions make little sense without session or its surrogate. In contrast to RBAC, the analyzed middleware technologies are request-oriented. Authorization decisions are based on the credentials associated with the request. As a result, subject's roles can be either pushed, as in CORBA, by CSS during connection establishment, or pulled, as in EJB and COM+, by the TSS from its (possibly local) data store. Moreover, since in COM+ roles are specific to each COM+ application and are pulled by TSS, remote clients have no control over the selection of the roles activated when they access individual COM+ applications, making user-driven role activation/deactivation impractical. It seems that making ANSI RBAC session-oriented prevents request-oriented (e.g., COM+, EJB and CORBA) and possibly other systems from supporting the standard in full. This is why we second suggestion by Li et al. [68] to make session-specific functionality an optional component of the standard.

Another caveat with support for ANSI RBAC in COM+ becomes apparent when one considers a multi-host COM+ deployment. Unless COM+ applications are managed through Active Directory, administrative functions would have to be performed on host-by-host basis, making the process error-prone. Also such functions as *UserPermissions* would require enumeration of all COM+ objects, which is not presently supported in multi-host deployments of COM+ applications.

In addition to the lack of support for the aforementioned functions, role hierarchy is not directly supported in COM+ or in the Microsoft Component Services GUI described in Section 2.2.2. Nonetheless, support for role hierarchy is still possible, thanks to the COMAdmin [36] library's APIs. A custom administrative application can be created to replace the Component Services GUI and provide support for role hierarchy. This application is required to maintain role hierarchy relations and manipulate various COM+ collections, such as the *UsersInRole* collection (see Fig. 8). For example, when a role hierarchy is introduced, the custom administrative application is required to manipulate the *RolesForComponent*, *RolesForInterface*, and the *RolesForMethod* COM+ collections to ensure that appropriate roles are added to these collections to reflect the fact that roles would inherit permissions based on the role hierarchy.

Static separation of duty constraints are not supported by COM+. However, the custom administrative application described above would allow for this support. The application would implement functions—such as *AssignUser*—in a manner that would ensure that static SoD constraints are met before a user is assigned to a certain role. On the other hand, dynamic SoD constraints may not be implementable since role activation and sessions are handled by Windows.

In summary, the role-based access control provided by COM+ falls short of supporting all ANSI RBAC required functions, although COM+ has better support than CORBA or EJB. The limitations that prevent full support are mainly due to the mismatch between session-oriented nature of RBAC and request-oriented architecture of COM+ and other commercial middleware that we analyzed elsewhere [62,63]. This mismatch calls into question the mandatory support for sessions and related functionality in ANSI RBAC systems. When it comes to multi-

Author's personal copy

host deployments of COM+ systems RBAC administration becomes problematic and the lack of support for enumerating COM+ objects across hosts becomes an impediment. While role hierarchies and separation of duty constraints are not directly supported, they can be, with the help of custom tools for administering COM+ systems.

7. Conclusion

Understanding middleware access control mechanisms is critical for protecting the resources of enterprise applications. In this paper, we described in detail the architecture of access control mechanisms in COM+ and defined a configuration of the COM+ protection system in precise and unambiguous terms of set theory. Based on this definition, we formalized the semantics of authorization decisions in COM+. We analyzed support for various ANSI RBAC functions in COM+, and illustrated our discussion with an example. We also showed how some of the ANSI RBAC functions can be supported using Windows and COM+ programming interfaces. Our result indicate that, as in CORBA and EJB, the required functions for creating and deleting sessions and for activating and deactivations roles in a session are not supported. This failure to support session-specific functions across three major commercial middleware technologies points to a systemic mismatch between session-oriented architecture of RBAC and real-world distributed systems.

References

[1] D.E. Bell, L.J. LaPadula, Secure computer systems: unified exposition and multics interpretation, Technical Report ESD-TR-75-306, MITRE, March 1975.
[2] B.W. Lampson, Protection, 5th Princeton Conference on Information Sciences and Systems, ACM Press, New York, NY, USA, 1971, p. 437, http://portal.acm.org/citation.cfm?id=775268.
[3] L. Notargiacomo, Role-based access control in oracle7 and trusted oracle7, the First ACM Workshop on Role-Based Access Control, ACM Press, Gaithersburg, Maryland, USA, 1995, pp. 65–69.
[4] J. Epstein, R. Sandhu, Netware 4 as an example of role-based access control, Proceedings of the First ACM Workshop on Role-Based Access Control, ACM Press, Gaithersburg, Maryland, USA, 1995, pp. 71–82.
[5] L. Giuri, Role-based access control in Java, Proceedings of the Third ACM Workshop on Role-Based Access Control, ACM Press, Fairfax, Virginia, USA, 1998, pp. 91–99.
[6] W.J. Meyers, RBAC emulation on trusted dg/ux, Proceedings of the Second ACM Work- shop on Role-Based Access Control, ACM Press, Fairfax, Virginia, USA, 1997, pp. 55–60.
[7] F. Zhang, X. Sheng, Y. Niu, F. Wang, H. Zhang, The research and scheme of RBAC using J2EE security mechanisms, in: R. Jain, B.B. Dingel, S. Komaki, S. Ovadia (Eds.), Broadband Access Communication Technologies, vol. 6390, SPIE, Boston, MA, USA, 2006, p. 63900L, doi:10.1117/12.685797, http://link.aip.org/link/?PSI/6390/63900L/1.
[8] V. Bindiganavale, J. Ouyang, Role based access control in enterprise application—security administration and user management, 2006 IEEE International Conference on Information Reuse and Integration, IEEE, IEEE Press, Waikoloa Village, HI, 2006, pp. 111–116, doi:10.1109/IRI.2006.252397, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4018474.
[9] J. Barkley, Implementing role-based access control using object technology, The First ACM Workshop on Role-Based Access Control, ACM Press, Fairfax, Virginia, USA, 1995, pp. 93–98, http://www.acm.org/pubs/citations/proceedings/commsec/270152/p293-barkley/.
[10] R.K. Wong, RBAC support in object-oriented role databases, Proceedings of The Second ACM Workshop on Role-Based Access Control, ACM Press, Fairfax, Virginia, USA, 1997, pp. 109–120.
[11] J. Barkley, A. Cincotta, Managing role/permission relationships using object access types, The Third ACM Workshop on Role-Based Access Control, ACM Press, Fairfax, Virginia, USA, 1998, pp. 73–80.
[12] R. Awischus, Role based access control with security administration manager (SAM), The Second ACM Workshop on Role-Based Access Control, ACM Press, Fairfax, Virginia, USA, 1997, pp. 61–68.
[13] ANSI, ANSI INCITS 359-2004 for Role Based Access Control, 2004.
[14] OMG, The common object request broker: architecture and specification, Specification formal/99-10-08, Object Management Group, 1999.
[15] R.J. Oberg, Understanding & Programming COM+: A Practical Guide to Windows 2000 DNA, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000 http://portal.acm.org/citation.cfm?id=345834&dl=ACM&coll=portal#.
[16] L.G. DeMichiel, L.Ü. Yalcinalp, S. Krishnan, Enterprise JavaBeans Specification, Version 2.0, Sun Microsystems, 2001.
[17] G. Eddon, The COM+ security model gets you out of the security programming business, Microsoft Systems Journal 11 (1999).
[18] OMG, Common Object Services Specification, Security Service Specification v1.8, 2002.
[19] B. Hartman, D.J. Flinn, K. Beznosov, Enterprise Security With EJB and CORBA, John Wiley & Sons, Inc., New York, 2001 http://konstantin.beznosov.net/professional/books/enterprise_security_with_EJB_and_CORBA.html.
[20] D. Basin, F. Rittinger, A formal analysis of the CORBA security service, ZB 2002: Formal Specification and Development in Z and B, LNCS 2272, Springer, 2002, pp. 330–349.
[21] D. Ferraiolo, R. Kuhn, Role-based access controls, Proceedings of the 15th NIST-NCSC National Computer Security Conference, National Institute of Standards and Technology/Na- tional Computer Security Center, Baltimore, MD, USA, 1992, pp. 554–563.
[22] R. Sandhu, E. Coyne, H. Feinstein, C. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[23] R. Sandhu, D. Ferraiolo, R. Kuhn, The NIST model for role-based access control: towards a unified standard, RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control, ACM, New York, NY, USA, 2000, pp. 47–63, http://doi.acm.org/10.1145/344287.344301.
[24] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, ACM Transactions on Information and System Security 4 (3) (2001) 224–274 http://ite.gmu.edu/list/journals/tissec/p224-ferraiolo.pdf.
[25] Microsoft, COM+ Administration Collections, 2008 http://msdn2.microsoft.com/en-us/library/ms687763(VS.85).aspx.
[26] G. Eddon, Inside COM+ Base Services, Microsoft Programming Series, Microsoft Press, 1999 http://www.amazon.com/exec/obidos/ASIN/0735607281/qid=953825136/sr=1-1/103-0762350-4861460.
[27] D. Box, Essential COM, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997 foreword By-Grady Booch and Foreword By-Charlie Kindel.
[28] I. Sommerville, Software Engineering8th ed., Addison Wesley, 2006.
[29] N. Brown, C. Kindel, Distributed component object model protocol (DCOM/1.0), Tech. Rep. draft-brown-dcom-v1-spec-03.txt, Microsoft Corporation, January 1998, http://www.lisp-p.org/nmcom/draft-brown-dcom-v1-spec-03.html.
[30] TOG, DCE 1.1: Remote Procedure Call, The Open Group, catalog number c706 Edition (August 1997).
[31] C. Szyperski, D. Gruntz, S. Murer, Component Software: Beyond Object-Oriented Programming, Addison-Wesley Professional, 2002.
[32] Microsoft, Microsoft Interface Definition Language, 2005 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/midl/midl/midl_start_page.asp.
[33] S.P. Miller, B.C. Neuman, J.I. Schiller, J.H. Saltzer, Kerberos authentication and authorization system, Tech. rep. Massachusetts Institute of Technology, 1987, citeseer.ist.psu.edu/miller88kerbero.html.
[34] Microsoft, Microsoft NTLM, 2005 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthn/security/microsoft_ntlm.asp.
[35] C. Adams, S. Lloyd, Understanding PKI: Concepts, Standards, and Deployment Considerations2nd ed., Addison Wesley Professional, 2002.
[36] Microsoft, Automating COM+ Administration, 2006 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cossdk/html/f302eb02-2ef5-42ee-a18f-59f7e60b38df.asp.
[37] Sun Microsystems Inc., RBAC in the Solaris™ operating environment, http://www.sun.com/software/whitepapers/wp-rbac/wp-rbac.pdf, white Paper, 2000 http://www.sun.com/software/whitepapers/wp-rbac/wp-rbac.pdf.
[38] R. Sandhu, G.-J. Ahn, Decentralized group hierarchies in UNIX: an experiment and lessons learned, Proc. 21st NIST-NCSC National Information Systems Security Conference, National Institute of Standards and Technology/National Computer Security Center, Arlington, Virginia, USA, 1998, pp. 486–502, http://citeseer.ist.psu.edu/321354.html.
[39] G.-J. Ahn, R. Sandhu, Decentralized user group assignment in Windows NT, The Journal of Systems and Software 56 (1) (2001) 39–49.
[40] G. Faden, RBAC in UNIX administration, RBAC '99: Proceedings of the Fourth ACM Workshop on Role-Based Access Control, ACM Press, New York, NY, USA, 1999, pp. 95–101, http://doi.acm.org/10.1145/319171.319180.
[41] T.M. Chalfant, Role based access control and secure shell — a closer look at two Solaris™operating environment security features, Tech. rep., Sun BluePrints™On-Line, June 2003, http://www.sun.com/blueprints/0603/817-3062.pdf.
[42] S. Tran, M. Mohan, Security Information Management Challenges and Solutions, http://www.ibm.com/developerworks/db2/library/techarticle/dm-0607tran/index.html, 2006 http://www.ibm.com/developerworks/db2/library/techarticle/dm-0607tran/index.html.
[43] MySQL AB, MySQL, http://www.mysql.com, 2007 http://www.mysql.com.
[44] C. Ramaswamy, R. Sandhu, Role-based access control features in commercial database management systems, Proc. 21st NIST-NCSC National Information Systems Security Conference, National Institute of Standards and Technology/National Computer Security Center, Arling- ton, VA, USA, 1998, pp. 503–511, http://citeseer.ist.psu.edu/ramaswamy98rolebased.html.
[45] IBM, IBM Informix Dynamic Server Administrator's Guide, Informix Dynamic Server 10.0; Document ID: G251-2267-02, December 2005 http://www-306.ibm.com/software/data/informix/pubs/library/ids_100.html.
[46] Sybase Inc., System Administration Guide: Volume 1 — Adaptive Server Enterprise 15.0, document ID: DC31654-01-1500-02, October 2005 http://infocenter.sybase.com/help/topic/com.sybase.help.ase_15.0.sag1/sag1.pdf.
[47] R. Baylis, P. Lane, D. Lorentz, Oracle Database Administrator's Guide, 10g Release 1 (10.1), December 2003 http://otn.oracle.com/pls/db10g/db10g.homepage.
[48] K. Gutzmann, Access control and session management in the HTTP environment, IEEE Internet Computing 5 (1) (2001) 26–35 http://dx.doi.org/10.1109/4236.895139.
[49] J.S. Park, R. Sandhu, G.-J. Ahn, Role-based access control on the web, ACM Transactions on Information and System Security (TISSEC) 4 (1) (2001) 37–71.

Author's personal copy

[50] R. Robles, M.-K. Choi, S.-S. Yeo, T. hoon Kim, Application of role-based access control for web environment, Ubiquitous Multimedia Computing, UMC '08. International Symposium on (2008), 2008, pp. 171–174, doi:10.1109/UMC.2008.41.

[51] L.S. Bartz, hyperDRIVE: leveraging LDAP to implement RBAC on the web, Proceedings of the Workshop on Role-based Access Control, ACM Press, New York, NY, 1997, pp. 69–74, http://doi.acm.org/10.1145/266741.266759.

[52] D.F. Ferraiolo, J.F. Barkley, D.R. Kuhn, A role-based access control model and reference implementation within a corporate intranet, ACM Transactions on Information and System Security (TISSEC) 2 (1) (1999) 34–64.

[53] D.W. Chadwick, A. Otenko, The PERMIS X.509 role based privilege management infrastructure, SACMAT '02: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, ACM Press, New York, NY, USA, 2002, pp. 135–140, http://doi.acm.org/10.1145/507711.507732.

[54] W. Zhou, C. Meinel, Implement role based access control with attribute certificates, The 6th International Conference on Advanced Communication Technology (ICACT2004), vol. 1, National Computerization Agency, Electronics and Telecommunications Research Institute, Korea, Korea, 2004, pp. 536–541.

[55] M. Wahl, T. Howes, S. Kille, RFC 2251: Lightweight Directory Access Protocol, vol. 3, 1997 http://rfc.net/rfc2251.html.

[56] L. Giuri, Role-based access control on the Web using Java, Proceedings of the Fourth ACM Workshop on Role-based Access Control, ACM Press, New York, NY, USA, 1999, pp. 11–18, http://doi.acm.org/10.1145/319171.319173.

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

[58] Microsoft, DCOM Architecture, 1998 http://www.microsoft.com/NTServer/app-service/techdetails/prodarch/DCOM/2_DCOMArchitecture.asp.

[59] C. Westphall, J. Fraga, A large-scale system authorization scheme proposal integrating Java, CORBA and web security models and a discretionary prototype, Latin American Network Operations and Management Symposium, IEEE Press, Rio de Janeiro, Brazil, 1999, pp. 14–25.

[60] R.R. Obelheiro, J.S. Fraga, Role-based access control for CORBA distributed object systems, Proceedings of the IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002), IEEE, Washington, DC, USA, 2002, p. 53.

[61] C.M. Westphall, J. da Silva Fraga, M.S. Wangham, R.R. Obelheiro, L.C. Lung, PoliCap— proposal, development and evaluation of a policy service and capabilities for CORBA Security, SEC '02: Proceedings of the IFIP TC11 17th International Conference on Information Security, Kluwer, B.V., Deventer, The Netherlands, 2002, pp. 263–274.

[62] K. Beznosov, W. Darwish, Support for ANSI RBAC in CORBA, Technical Report LERSSE- TR-2007-01, accessible from http://lersse-dl.ece.ubc.ca/search.py?recid=129, Laboratory for Education and Research in Secure Systems Engineering, University of British Columbia, July 27 2007, http://lersse-dl.ece.ubc.ca.

[63] W. Darwish, K. Beznosov, Support for ANSI RBAC in EJB, Technical Report LERSSE-TR-2009-34, accessible from http://lersse-dl.ece.ubc.ca, Laboratory for Education and Research in Secure Systems Engineering, University of British Columbia, January 21 2009, http://lersse-dl.ece.ubc.ca.

[64] J. Lowy, Windows XP: Make Your Components More Robust with COM+ 1.5 Innovations, August 2001 http://msdn.microsoft.com/msdnmag/issues/01/08/ComXP/default.aspx.

[65] Microsoft, Network Management, 2006 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/network_management.asp.

[66] P. Mockapetris, Domain Names — Concepts and Facilities, 1987 http://www.ietf.org/rfc/rfc1034.txt.

[67] N.W. Group, Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods, 1987 http://www.ietf.org/rfc/rfc1001.txt.

[68] N. Li, J.-W. Byun, E. Bertino, A Critique of the ANSI Standard on Role Based Access Control, CERIAS and Department of Computer Science, March 3 2006 http://www.cs.purdue.edu/homes/ninghui/papers/aboutRBACStandard.pdf.