1998 International Computer Symposium
Workshop on Cryptology and Information Security
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

# SECURITY ARCHITECTURE OF DCOM AND ITS INTEGRATION WITH RBAC

*Gail-Joon Ahn and Ravi Sandhu*
Information and Software Engineering Department, MS 4A4
George Mason University
Fairfax, VA 22030, U.S.A.
{gahn,sandhu}@isse.gmu.edu

## ABSTRACT

The explosive growth of the Web, the increasing popularity of PCs and the advances in high-speed network access have brought distributed computing into the mainstream. To simplify network programming and to realize component-based software architecture, distributed object models have emerged as standards. One of those models is DCOM (Distributed Component Object Model) which is a protocol that enables software components to communicate directly over a network in a reliable, and efficient manner. In this paper, we investigate an aspect of DCOM concerning software architecture and security mechanism. Also, we describe the concept of role-based access control (RBAC) which began with multi-user and multi-application on-line systems pioneered in the 1970s. And we investigate how we can enforce the role-based access control as a security provider within DCOM, specially in access security policy.

## 1 INTRODUCTION

Distributed applications introduce new design and deployment issues. For this added complexity to be worthwhile, there has to be a significant payback. Some applications are inherently distributed. Multiuser games and teleconferencing applications are examples of such applications. For these, the benefits of a robust infrastructure for distributed computing are obvious. Many other applications are also distributed, in the sense that they have at least two components running on different machines. But because these applications were not designed to be distributed, they are limited in scalability and ease of deployment. Any kind of workflow or groupware application, most client/server applications, and even some desktop productivity applications essentially control the way their users communicate and cooperate. Viewing of these applications as distributed applications and running the right components in the right places benefits the user and optimizes the use of network and computer resources. The application designed with distribution in mind can accommodate different clients with different capabilities by running components on the client side when possible and running them on the server side when necessary.

DCOM is an extension of the Component Object Model (COM). COM defines how components and their clients interact. This interaction is defined such that the client and the component can connect without the need of any intermediary system component. The client calls methods in the component without any overhead whatsoever. This study will look at the DCOM architecture and DCOM's security mechanism. DCOM provides an extremely efficient default security mechanism that lets developers write distributed applications without having to worry about security at all. Any security provider supported by windows NT can be used with DCOM's security mechanism.

The concept of role-based access control (RBAC) began with multi-user and multi-application on-line systems pioneered in the 1970s. The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This simplifies management of permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Role can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Several researchers have showed that RBAC can be accommodated in current systems, such as Oracle, Unix.

But DCOM does not have role concept but rather than groups. With mapping and analysis, it is possible for RBAC to be applied to DCOM security mechanism with RBAC's advantages, which can satisfy DCOM's access security policy and achieve ease of administration.

This paper begins with the description of the DCOM model in section 2. In section 3, we discuss the RBAC model following by section 4 which includes integration between RBAC model and DCOM. Section 5 concludes the paper.

## 2 DISTRIBUTED COMPONENT OBJECT MODEL

In programming and engineering disciplines, a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions. In programming design, a system is divided into components that in turn are made up of modules. In object-oriented programming and distributed object technology, a component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Examples of a component include: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be deployed on different servers in a network and communicate with each other for needed services [4]. A component runs within a context called a container. Examples of containers include pages on a Web site, Web browsers, and word processors.

COM (Component Object Model) is Microsoft's framework for developing and supporting program component objects. The Component Object Model provides a set of interfaces allowing clients and servers to communicate within the same computer (running a Windows 95 or NT system). It is aimed at providing similar capabilities to those defined in CORBA (Common Object Request Broker Architecture), the framework for the interoperation of distributed objects in a network. Whereas OLE provides services for the compound document that users see on their display, COM provides the underlying services of interface negotiation, life cycle management (determining when an object can be removed from a system), licensing, and event services (putting one object into service as the result of an event that has happened to another object).

DCOM (Distributed Component Object Model) is a protocol that enables software components to communicate directly over a network in a reliable, and efficient manner. At the same time it is a program interface in which client program objects can request services from server program objects on other computers in a network.

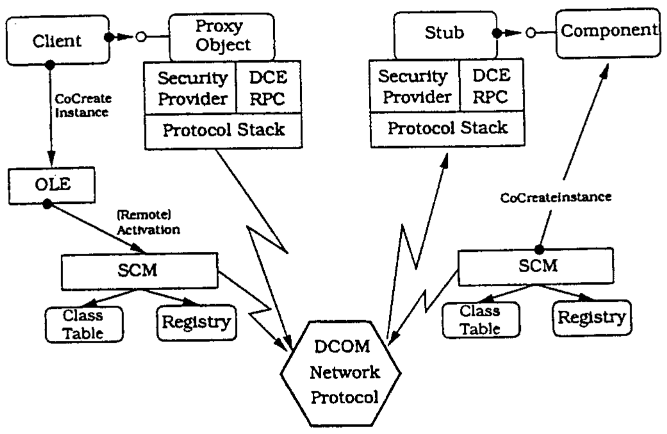

Figure 1: DCOM Architecture

For example, one can create a page for a Web site that contains a script or program that can be processed (before being sent to a requesting user) not on the Web site server but on another, more specialized server in the network. Using DCOM interfaces, the Web server site program (now acting as a client object) can forward a Remote Procedure Call (RPC) to the specialized server object, which provides the necessary processing and returns the result to the Web server site. This result is passed on to the Web page viewer.

Next, we will look at DCOM's architecture and the processing steps DCOM takes. We also discuss its security mechanism and policy.

### 2.1 DCOM's architecture

A client that needs to communicate with a component in another process cannot call the component directly, but has to use some form of inter-process communication provided by the operating system. COM provides this communication in a completely transparent fashion: it intercepts calls from the client and forwards them to the component in another process. When client and component reside on different machines, DCOM simply replaces the local inter-process communication with a network protocol. Neither the client nor the component are aware that the wire that connects them has just become a little longer. Figure 1 shows the overall DCOM architecture: the COM run-time provides object-oriented services to clients and components and uses RPC and the security provider to generate standard network packets that conform to the DCOM wire-

1998 International Computer Symposium
Workshop on Cryptology and Information Security
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

| **Client Initialization** |
|---|
| The client calls CoInitialize which initializes the COM library for use. Next, the client calls COM library's CoCreateInstance, passing the GUID for the desired class (CLSID) and an array of desired interface IDs (IIDs). |
| **Server Activation (Client side)** |
| The call to CoCreateInstance is passed on to the Service Control Manager (SCM) on the client machine which looks up the Class ID in the local Class Table to see if the server is already running. If not the SCM will look in the Registry using the Class ID to find the type and location of the server. |
| **Server Activation (Server Side)** |
| Using the server machine address, the client SCM establishes an RPC link with the server machine's SCM, passing it the desired CLSID and IID(s). The server SCM checks its Class Table to see if the server is running, and if it isn't, it look up the CLSID in the Registry to find out what command needs to be excuted to start the server, and executes it. When a COM server starts up, the first thing it calls is CoInitialize. The second thing it calls is CoRegistryClassObject, passing the implemented Class ID and a pointer to the server program's Class Factory. This effectively advertises the server program to DCOM, adding its entry into the Class Table. |
| **The Class Factory** |
| Whenever a client requests a new instance of a server calss, the SCM calls a method called CreateInstance on the server program's Class Factory. This is the common gateway used by all clients. CreateInstance then instantiates the object and passes the pointer to the object's IUnknown interface back to the SCM. |
| **Multiple Query Interface (MQI)** |
| Once the SCM has a pointer to the object's IUnknown interface, it can use Query-Interface to request pointers to other interfaces. |
| **Proxy/Stub Loading** |
| Data passed back between client and server has to be packaged into RPC packets. This process is called marshalling, and is performed by the Proxy/Stub DLL(s). Using the IID(s) returned from the MQI step, the SCMs on each machine interrogate the Registry for the DLL(s) that need to be loaded into the client's and server's process spaces. Interface Pointers are then handed back to the client that issued the CoCreateInstance call. |
| **Method Call** |
| Using the Interface Pointer, the client calls the server method. |

Table 1: DCOM steps

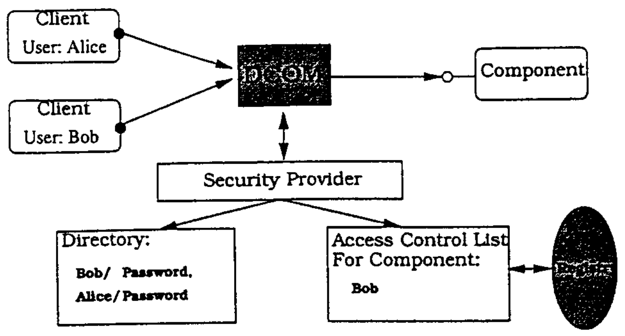

Figure 2: Security Mechanism

protocol standard.

Table 1 describes how DCOM works [6]. In summary, the client program calls CoInitialize, then CoCreateInstance passing the CLSID and IID(s). The client receives back Interface Pointer(s), and then calls the desired method. The server program implements the interface as a class, and has only two extra calls to CoInitialize and CoRegisterClassObject.

## 2.2 DCOM's security mechanism

Different platforms use different security providers, and many platforms even support multiple security providers for different usage scenarios or for interoperability with other platforms. DCOM and RPC are designed in such a way that they can simultaneously accommodate multiple security providers. All these security providers provide a means of identifying a security principal, a means of authenticating a security principal, and a central authority that manages security principals and their keys. If a client want to access a secured resource, it passes its security identity and some form of authenticating data to the resource and the resource asks the security provider to authenticate the client. Security providers typically use low-level custom protocols to interact with clients and protected resources.

Using the network for distributing an application is challenging not only because of the physical limitations. It also raises new issues related to security between and among clients and components. Since many operations are now physically accessible by anyone with access to the network, access to these operations has to be restricted at a higher level. Without security support from the distributed development platform, each application would be forced to implement its own security mechanisms. A typical mechanism would involve passing some kind of username and password—usually encrypted—to some kind of logon method. The application would validate these credentials against a user database or directory and return some dynamic identifier for use in future method class. On each subsequent call to a secure method, the clients would have to pass this security identifier. Each application would have to store and manage a list of usernames and passwords, protect the user directory against unauthorized access, and manage changes to passwords, as well as dealing with the security hazard of sending passwords over the network. A distributed platform must thus provide a security framework to safely distinguish different clients or different groups of clients so that the system or the application has a way of knowing who is trying to perform an operation on a component. DCOM uses the extensible security framework provided by Windows NT. Widows NT provides a solid set of built-in security providers that support multiple identification and authentication mechanisms, from traditional trusted-domain security models to noncentrally managed, massively scaling public-key security mechanisms. A central part of the security framework is a user directory, which stores the necessary information to validate a user's credentials. DCOM can make distributed applications secure without any security-specific coding or design in either the client or the component. Just as the DCOM programming model hides a component's location, it also hides the security requirements of a component. The same binary code that works in a single-machine environment, where security may be of no concern, can be used in a distributed environment in a secure fashion.

DCOM's default security mechanism is illustrated in figure 2. DCOM achieves security transparency by letting developers and administrators configure the security settings for each component. Just as the Windows NT File System lets administrators set access control lists (ACLs) for files and directories, DCOM stores Access Control Lists for components. These lists simply indicate which users or groups of users have the right to access a component of a certain class. These lists can easily be configured using the DCOM configuration tool. Whenever a client calls a method or creates an instance of a component, DCOM obtains the client's current username associated with the current process. Windows NT guarantees that this user credential is authentic. DCOM then passes the username to the machine or process where the component is running. DCOM on the component's machine then validates the username again using authentication mechanism and checks the access control list for the component. If the client's username is not included in this list (either directly or indirectly as a member of a groups of users), DCOM simply rejects the call before the component is ever involved. This default security mechanism is completely transparent to both the client and the component and is highly optimized. It is based on the Windows NT security framework, which is probably one of the most heavily used parts of the Windows NT operating system: on each and every access to a file or even to a thread-synchronization primitive like an event or semaphore.

## 2.3 DCOM's security policy

DCOM distinguishes between four fundamental aspects of security [7]:

- **Access security: protecting the object**
  Which security principals are allowed to call an object?
  The most obvious security requirement on distributed applications is the need to protect objects against unauthorized access. Sometimes only authorized users are supposed to be able to connect to an object. In other cases, non-authenticated or unauthorized users might be allowed to connect to an object, but must be limited to certain areas of functionality. Current implementations of DCOM provide declarative access control on a per-process level.

- **Launch security: protecting the server machine**
  Which security principals are allowed to create a new object in a new process?
  Another related requirement on a distributed infrastructure is to maintain control over who can create objects. Since all COM objects of a machine are potentially accessible via DCOM, it is critical to prevent unauthorized users from creating instances of these objects.

- **Security identity: controlling the object**
  What is the security principal of the object itself?
  Another aspect of distributed security is that of controlling the objects themselves. Since an object performs operations on behalf of arbitrary callers, it is often necessary to limit the capabilities of the object itself. One obvious approach is that of making the object assume the identity of the caller. Whatever action the object performs is limited by the caller's privileges. Although managing access can be simplified by using user groups, it is often simpler to have the object itself run under a dedicated security identity, independent of the security identity of the current caller.

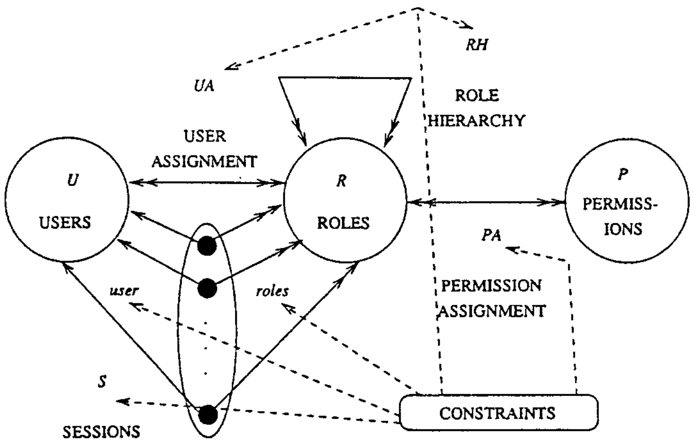

Figure 3: RBAC Model

- **Connection policy**

  **Integrity**: can messages be altered?

  **Privacy**: can messages be intercepted by others?

  **Authentication**: can the object find out or even assume the identity of the caller?

The scope of this paper will be within the first security policy, *access security*. The following sections will show how to simulate this security issue with RBAC.

## 3 OVERVIEW OF RBAC MODEL

RBAC is an alternative access control policy to mandatory and discretionary access control. As MAC is used in the classical defense arena, the policy of access is based on the classification of objects such as top-secret level. But RBAC policy is based on the role of the subjects and can specify security policy in a way that maps to an organization's structure.

A general family of RBAC models called RBAC96 was defined by Sandhu et al [1]. Figure 3 illustrates the most general model in this family. Motivation and discussion about various design decisions made in developing this family of models is given in [1, 2]. Also, there are variations regarding distributed systems [5].

The figure 3 shows (regular) roles and permissions that regulate access to data and resources. Intuitively, a user is a human being or an autonomous agent, a role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role, and a permission is an approval of a particular mode of access to one or more objects in the system or some privilege to carry out specified actions. Roles are organized in a partial order $\geq$, so that if $x \geq y$ then role $x$ inherits the permissions of role $y$. Members of $x$ are also implicitly members of $y$. In such cases, we say $x$ is senior to $y$. Each session relates one user to possibly many roles. The idea is that a user establishes a session and activates some subset of roles that he or she is a member of (directly or indirectly by means of the role hierarchy). The RBAC model has the following components and these components are formalized from the above discussions.

- $U$ is a set of users,
- $R$ is disjoint sets of roles and administrative roles respectively,
- $P$ is disjoint sets of permissions and administrative permissions,
- $UA \subseteq U \times R$, is a many-to-many user to role assignment relation,
- $PA \subseteq P \times R$ is many-to-many permission to role assignment relations,
- $RH \subseteq R \times R$ is partially ordered role hierarchies (written as $\geq$ in infix notation),
- $S$ is a set of sessions,
- $user : S \rightarrow U$, is a function mapping each session $s_i$ to the single user $user(s_i)$ and is constant for the session's lifetime,
- $roles : S \rightarrow 2^R$ is a function mapping each session $s_i$ to a set of roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ (which can change with time) so that session $s_i$ has the permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA]\}$, and
- there is a collection of constraints stipulating which values of various components of the RBAC model are allowed or forbidden.

A user can be a member of many roles and a role can have many users. Similarly, a role can have many permissions and the same permissions can be assigned to many roles. Each session relates one user to possibly many roles. Intuitively, a user establishes a session during which the user activates some subset of roles that

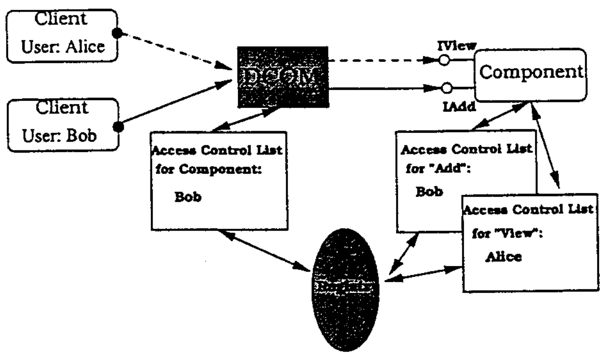
Client
User: Alice
Client
User: Bob
IView
IAdd
Component
Access Control List for Component: Bob
Access Control List for "Add": Bob
Access Control List for "View": Alice

Figure 4: Security using Registry Key

he or she is a member of the permissions available to the users are the union of permissions from all roles activates in that session. Each session is associated with a single user. This association remains constant for the life of a session. A user may have multiple sessions open at the same time, each in a different window on the workstation screen for instance. Each session may have a different combination of active roles. The concept of a session equates to the traditional notation of a subject in access control. A subject is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions active at the same time.

## 4 INTEGRATION WITH RBAC

In this section we outline one approach to enforcing RBAC in DCOM. For some applications, a single component-wide access control list which is described in section 2.2 is not sufficient. Some methods in a component may be accessible only to certain users. For example, an accounting business component may have a method for registering new transactions and another method for retrieving existing transactions. Only members (such as Bob) of the accounting department (user group "Accounting") should be able to add new transactions, while only members (such as Alice) of transaction management (user group "Transaction") should be able to view the transactions. And members of top management (user group "Top Management") should be able to add and view the transaction. How can an application use DCOM security to implement the selective security required in this example?

We can approach this example with programmatic control using DCOM [8]. This approach is shown in figure 4. When a method call comes in, the component

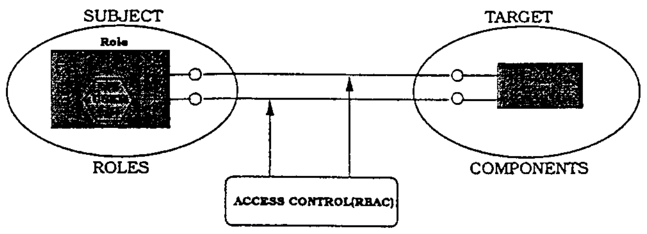

Figure 5: Conceptual abstraction of RBAC

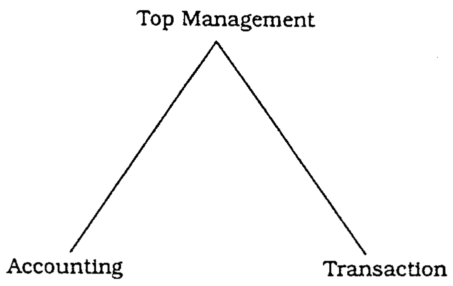

Figure 6: An example of role hierarchy

asks DCOM to impersonate the client. After this, the called thread can perform only those operations on secured objects, that the client is permitted to perform. The component can then try to access a secured object, such as a registry key, that has an Access Control List on it. If this access fails, the client was not contained in the ACL, and the component rejects the method call. By choosing different registry keys according to the method that is being called, the component can provide selective security in a simple way.

We can simulate RBAC in DCOM along the lines of this example[1]. Figure 5 shows a conceptual abstraction of a role-based access control model. With this abstraction we can see that RBAC can be inserted into DCOM architecture (in figure 1) as a part of security provider.

An approach to mapping the above model to design mechanisms is to consider the role-based access control model as a security-specific abstraction of the software architecture of the system.

In order to use RBAC model, we should accommodate role-hierarchies. Roles would map to NT user group(s) (which do not support hierarchies). For example, user group "Accounting" would map to role "*Accounting*". We can represent the role-relationship as role-hierarchy. The role hierarchy of this example would be as shown in figure 6. The *Accounting* role can have permission to add new transactions, while only the *Transaction* role can have permission to view the transactions. The *Top Management* role is senior to *Accounting* and *Transaction* and thereby inherits all permissions from junior roles.

In figure 4, the DCOM checks the ACL of component and then the component checks the ACL of methods. Whenever the component is accessed the ACL of the component should be checked. Instead of doing this two-step process, we can have a unified checking mechanism using RBAC as shown in figure 7. In the role hierarchy checking step, the client's role should be checked. After that the role permission check decides whether the client can have permission(s) to access the component according to given role(s)[2]. For example, assume that a client Chris has a role *Top Management* in figure 6 and tries to access an accounting business component. During role hierarchy check, we can know Chris's role memberships {*Accounting, Transaction, Top Management*}. After the role permission check, he can have all permissions from junior roles such as *Accounting* and *Transaction* using Registry Key. It means Chris can add new transactions and also view the transaction. Let's consider that Bob has a role *Accounting* in figure 6 and tries to access an accounting business component. We can simply know that Bob can only add new transactions but he can not view the transaction. Using an approach with RBAC we can have the same result as the previous approach illustrated in figure 4. The RBAC approach can also reduce the interactions between a component and secure object such as Registry. And we can have a separate security mechanism because the integrated RBAC can only access the secure object such as Registry.

Integrating with RBAC, we just showed that we can simplify access control in DCOM. In further study, we will investigate the details how RBAC can be achieved in the end-to-end mechanism of DCOM.

## 5 CONCLUSION

In this paper, we have described the architecture and security mechanism of DCOM. Also, we briefly looked at the RBAC model as a security specification model. Finally, we have shown that RBAC can be accommodated in DCOM. Also we can see that the adoption of Role-Based Access Control as a part of security provider can simplify access control and achieve ease of administra-

[1]We assume that the assignment is done by [3, 9], including the simulation of role-hierarchies.

[2]These permissions can easily be configured using DCOM configuration tool (addressed in section 2.2).

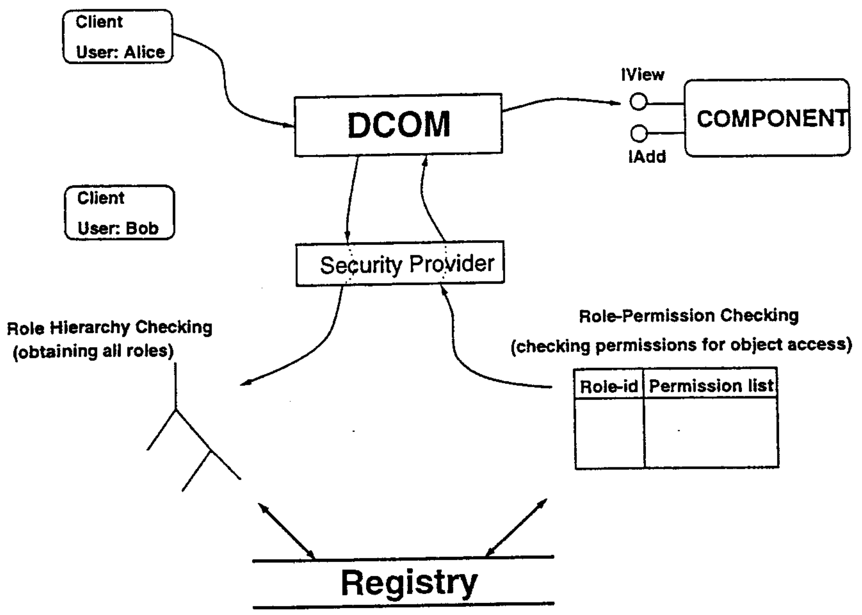

Figure 7: Integration with RBAC

tion. This framework was just based on DCOM's access security policy. In the future work, we also would investigate whether DCOM has sufficient flexibility to accommodate administrative access control models such as URA97 [10].

## References

[1] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[2] Ravi Sandhu. Rationale for the RBAC96 family of access control models. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1996.

[3] Ravi Sandhu and Gail J. Ahn. Decentralized Group Hierarchies in UNIX: An Experiment and Lessons Learned. In *Proceedings of 21st NIST-NCSC National Information Security Conference*, 1998.

[4] M. Moriconi and et al. Secure Software Architectures. In *Proceedings of IEEE symposium on Security and Privacy*, Oakland, May 1997.

[5] Nicholas Yialelis,Emil Lupu, and Morris Sloman. Role-Based Security for Distributed Object Systems. In *Proceedings of the IEEE Fifth Workshops on Enabling Technology: Infrastructure for Collaborative Enterprise*, Stanford, June 1996.

[6] Richard Grimes. Professional DCOM Programming. WROX Press Ltd.

[7] DCOM Architecture. *Microsoft Professional Developers Conference*. September, 1997.

[8] Microsoft Windows NT server DCOM Technical Overview White Paper. *http://www.microsoft.com/com/dcom95/*.

[9] Ravi Sandhu and Gail J. Ahn. Group Hierarchies with Decentralized User Assignment in Windows NT. In *Proceedings of IASTED Conference on Software Engineering*, October, 1998.

[10] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects*. North-Holland, 1997.