An Evolution of DCE Authorization Services

One of the strengths of the Open Software Foundation's Distributed Computing Environment is that it allows developers to consider authentication, authorization, privacy, and integrity early in the design of a client/server application. The HP implementation evolves what DCE offers to make it easier for server developers to use.

by Deborah L. Caswell

In the Open Software Foundation's Distributed Computing Environment (DCE),[1,2] services are provided by server processes. They are accessed on behalf of users by client processes often residing on a different computer. Servers need a way to ascertain whether or not the user has a right to obtain the service requested. For example, a banking service accessed through an automated teller machine has to have a way to know whether the requester is allowed to withdraw money from the account. A medical patient record service has to be able to know both who you are and what rights you should have with respect to a patient's record. A policy can be implemented such that only the patient or the legal guardian of the patient can read the record, but doctors and nurses can have read and write access to the record.

The process of determining whether or not a user has permission to perform an operation is called authorization. It is common to separate the authorization policy from the authorization mechanism. Authorization policy dictates who has permission to perform which operations on which objects. The mechanism is the general-purpose code that enforces whatever policy is specified. In DCE, the encoding of the authorization policy is stored in an access control list (ACL). Every object that is managed by a server such as a bank account or a patient record has associated with it an ACL that dictates which clients can invoke each operation defined for the object.

For example, to encode the policy that the owner of the bank account can deposit and withdraw money from the account and change the mailing address on the account, but only a bank teller may close the account, an ACL on a bank account owned by client Mary might look like:

```
user:Mary:DWM
group:teller:C
```

where `D` stands for permission to deposit, `W` for permission to withdraw, `M` for permission to change the mailing address, and `C` for permission to close the account.

Each application is free to define and name its own set of permissions. The `D`, `W`, and `C` permissions used in the example above are not used by every server. An application in which the `D` (deposit) permission makes sense could choose to name it as the "+" permission. Also, many applications will not have a deposit operation at all. Therefore, the interpretation of an ACL depends on the set of permissions defined by the server that uses it.

The first part of this paper describes the specifications and authorization mechanisms (code) offered in DCE that support the development of authorization services. The second part describes our efforts to supplement what DCE offers to make it easier for the server developer to use authorization services. The ACL functionality described here pertains to DCE releases before OSF DCE 1.1 and HP DCE 1.4.

Authorization Based on Access Control Lists

Fig. 1 shows the client/server modules required for an ACL authorization scheme used in a hypothetical bank application that was implemented using DCE. To understand the

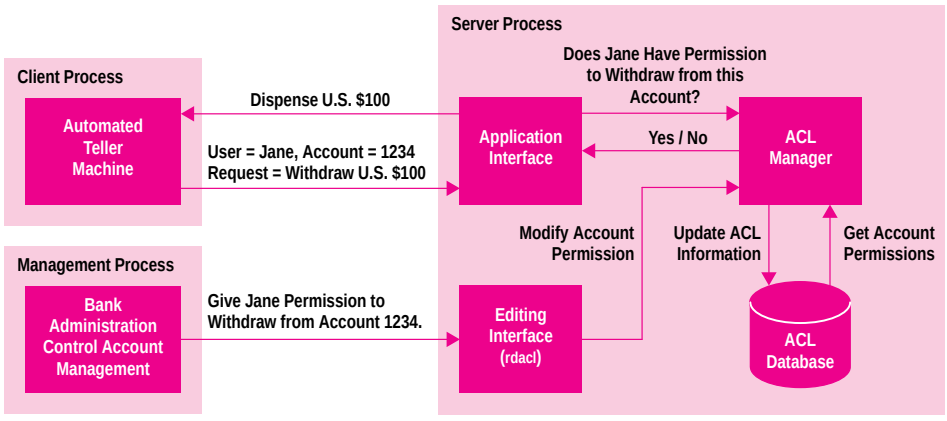

Fig. 1. Flow of information in the bank automatic teller machine example.

interactions between these modules consider the following scenario. Jane makes a request to withdraw U.S. $100.00 from her account number 1234. The application interface passes this information to the ACL manager asking for an authorization decision. The ACL manager retrieves the authorization policy for account 1234 from the ACL database and applies the policy to derive an answer. If Jane is authorized, the machine dispenses the money.

When Jane's account is first set up, a bank employee would use an administration tool (from the management process in Fig. 1) to give Jane permission to withdraw money from account 1234. The editing interface enables the ACL manager to change the policy. The ACL manager changes a policy by retrieving the current policy, modifying it, and writing it back to the ACL database.

ACL Database. A server that needs to authorize requests must have a way to store and retrieve the ACLs that describe the access rights to the objects the server manages. One application might want to store ACLs with the objects they protect and another might want a separate ACL database. Depending on the number of objects protected and access patterns, different database implementations would be optimal. For this reason, the requirements for an ACL storage system are likely to be very dependent on the type of application.

An Authorization Decision. When an application client makes a request of the application server, control is given to the manager routine that implements the desired operation. The manager routine needs to know what set of permissions or access rights the client must possess before servicing the request.

The manager routine must supply the client's identity (Jane), the name of the protected object (Account 1234), and the desired permissions (withdraw) to a routine that executes the standard ACL authorization algorithm. If the routine returns a positive result, the server will grant the client's request (dispense U.S. $100). Note that the authorization system depends on the validity of the client's identity. Authentication is a necessary prerequisite for authorization to be meaningful.

Standard Interface for Editing. Without a standard way of administering ACLs, each server developer would have to provide an ACL administration tool, and DCE administrators would have to learn a different tool for each server that uses authorization. To avoid that problem, a standard ACL editing interface is defined so that the same tool can interact with any service that implements the standard interface.

What DCE Provides

To meet the requirements for the ACL management scheme mentioned above, DCE provides code to support ACL management for some requirements and simply defines a standard interface without providing any code for other requirements. Fig. 2 shows the main components that provide DCE ACL support within the server executable.

Unforgeable Identities. DCE provides the run-time RPC (remote procedure call) mechanism, which provides the server process with information about the client making a request. Because of the authentication services provided in DCE, the client's identity is unforgeable so that the server need not worry about an impostor.

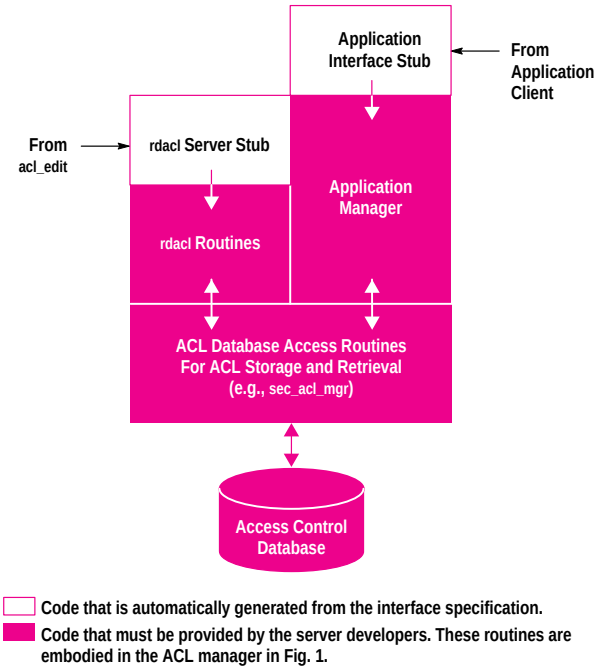

Fig. 2. Components that provide DCE ACL support in a server executable.

ACL Database. DCE suggests an interface to an ACL storage and retrieval subsystem called sec_acl_mgr. This interface is used within the server, and therefore is not mandatory or enforceable. DCE currently does not provide an implementation of this interface for use by application developers. Furthermore, it does not contain operations for adding and deleting ACLs, so even if the sec_acl_mgr interface is used, it would have to be supplemented by other ACL database access operations.

Authorization Decisions. DCE specifies a standard way of reaching an authorization decision given a client's identity, desired operation, and authorization policy encoding. The OSF DCE 1.0 distribution for application developers does not supply an implementation of this algorithm, requiring the server developer to write the authorization algorithm.

Standard Editing Interface. DCE provides a tool called acl_edit that an administrator can use to change the authorization policy used by any server that implements the standard rdacl interface even though each server might use a different set of permissions.

DCE defines the standard rdacl interface responsible for enabling modification of the authorization policy. The rdacl interface is used by acl_edit to access and modify ACL information. DCE does not provide an implementation of the rdacl interface. Without additional help from other sources, each server developer has to write rdacl routines that call the ACL database access routines. Servers that implement the rdacl interface can be administered by any client that uses the standard interface including the acl_edit tool mentioned above.

The rdacl interface does not support adding and deleting ACLs; it only addresses editing existing ACLs. For that reason, an ACL storage subsystem must be designed and implemented for an application that supports adding, modifying, retrieving, and deleting ACLs.

The rdacl operations listed below are described in the DCE reference manual.[3] They are listed here to give an idea of the size and functionality of the interface.

- rdacl_get_access: lists the permissions granted to a principal to operate on a particular object
- rdacl_get_manager_types: gets the list of databases in which the ACL resides
- rdacl_get_printstring: gets the user description for each permission
- rdacl_get_referral: gets a reference to the primary update site
- rdacl_lookup: gets the ACL for an object
- rdacl_replace: replaces the ACL for an object
- rdacl_test_access: returns true if the principal is authorized to perform the specified operation on an object
- rdacl_test_access_on_behalf: returns true if both the caller and a specified third-party principal are authorized to perform the specified operation on an object.

An implementation of these operations has to call the retrieve and modify operations of the ACL storage subsystem, invoke the authorization decision routine, and describe the permissions that are used in the ACLs for the particular implementation.

Component Relationships. Some of the boxes in Fig. 2 represent code that is automatically generated from the interface description, and other boxes represent code that must be supplied by server developers.

The modules on the right side of the block diagram in Fig. 2 represent the application-specific interfaces and code. The application interface stub is the code generated by the Interface Definition Language (IDL) compiler when given the application interface files. For example, if we have a bank account server, the application interface stubs would receive the call and direct it to the application manager. The application managers are the modules that implement the application server functionality. In our bank example, this is the code that implements the deposit and withdrawal operations.

On the left side of Fig. 2 is the code that is specific to ACL management of the DCE standard rdacl interface. The rdaclif (rdacl interface file) server stubs are generated by running the IDL compiler over the rdaclif.idl file which is delivered with the DCE product. The rdacl routines implement the operations defined in the rdacl interface. The bottom of Fig. 2 shows the ACL storage and retrieval code. The rdacl routines make calls to the storage layer either to get the ACLs that will be sent over the wire to a requesting client or to replace a new ACL received from an ACL administration tool. The database access routines must also implement the standard ACL checking authorization algorithm and a routine to compute the effective permissions of a client with respect to a specific object. The application managers call the database access layer to get an authorization decision. For example, the code that implements the withdrawal operation needs to first make sure that the client making the request is authorized to withdraw money from a particular account.

Although they do not interact directly with each other, the application manager routines and rdacl routines coexist within the same process and call common ACL manager routines.

Summary

DCE supports a server process's ability to make an authorization decision in several ways, but as shown in Fig. 2, there is a lot of code left for the server developer to write. Some of the required code, such as the authorization decision routine, can be reused in other applications because it is application independent. Other code, such as the storage subsystem, is more application-specific and might have to be developed for each new service.

Help for the Server Developer

This section describes three evolutionary steps that we took to supplement DCE's authorization support. The approach we took to each step is not novel. Each approach has value by itself in addition to being a stepping stone to a more sophisticated approach.

Note that although the outputs from each of these steps did not directly become products, they did form the basis for HP Object-Oriented DCE (HP OODCE). HP OODCE is briefly described later in this article and completely described in the article on page 55.

Sample Applications. The first step was simply to provide an example of server code that performs ACL management. The application acl_manager is one of a set of sample applications written to demonstrate the use of various DCE facilities. These sample applications are a valuable learning tool and are also useful for cutting and pasting working code into a real-world application.

The acl_manager is based on the ACL manager reference implementation distributed with DCE source code. The sample application uses a static table of ACLs, and there is no operation for adding or deleting ACLs and no general storage manager. However, acl_edit can interact with this primitive ACL manager to view or modify the ACL for one of these static objects .

The acl_manager includes a description of how to tailor the code to one's own application server and provides more background on how ACL management works than is available in the DCE manual set.

Another sample application, the phone database, demonstrates the use of an ACL manager inside an application. This more complex sample application demonstrates how application interfaces and the ACL management interface coexist within the same server and how they interact. The phone database application uses an in-memory binary tree storage facility with a simple checkpoint facility for committing changes to stable storage. The persistent representation of ACLs can be modified by an editor for bulk input. At startup, the server parses and interprets this file.

As mentioned before, in addition to being a valuable learning tool, the sample applications provide reuse of code and ideas at the source-code level.

Common ACL Management Module Interface. Cutting a sample application and pasting it into a new application with an

understanding of how it needs to be modified is surely better than starting from scratch. Reuse through a code library is better yet. The problem was how to provide a single library for ACL management when so much of it is application-specific. There is so much flexibility in how ACLs are managed. We wondered if it were possible to anticipate what most developers would need and if we would be able to satisfy those needs by creating a general-purpose library.

The first task was to partition the aspects of ACL management into those that are application-specific and those that are application independent. The application independent portion would be provided as library routines. Our approach to the application-specific portions was threefold:

- Limit the flexibility by providing routines that would be sufficient for most developers. For example, although DCE allows a server to implement more than 32 permissions, limiting support to 32 or less simplified the design considerably.
- Parameterize routines such that their behavior can be determined when the library is initialized at startup. For example, each application defines its own set of permissions. A table of permissions can be downloaded into the library rather than hard-coded into the library routines.
- Identify a well-defined interface to the storage and retrieval routines. As mentioned earlier, the storage requirements are the one aspect of ACL management that will vary the most among applications. By partitioning the functionality in this way, customers with special storage needs can write their own ACL storage management, and provided that they conform to the published interface guidelines, would still be able to use the library for other ACL management functions.

Fig. 3 shows a different view of the ACL components depicted in Fig. 2. The application server component is not called out separately in Fig. 2. The server initialization code (server.c) is typically located in this component. The application server also contains the code that directs the DCE runtime code to start listening for incoming client requests.

The application manager component in Fig. 3 contains the same functionality as the application manager component shown in Fig. 2.

The ACL manager component in Fig. 3 represents the code needed to support the rdacl interface, the ACL checking algorithm, the computing of effective permissions, and other general utilities. Basically, the ACL manager contains all the ACL code that is independent of how an ACL is stored within a database. It also encapsulates the implementation of the ACL structure itself. In other words, if the data structure that represents an ACL were to change, only the ACL manager component would need to be rewritten to accommodate the changes.

The ACL storage manager contains the ACL database access routines and the ACL database. The ACL storage manager can manage ACL storage in memory, on disk, or a hybrid of the two.

The circled numbers in Fig. 3 correspond to the following interactions between ACL manager components.

1. The application server must call the ACL manager to initialize its internal data structures and to download application-specific information such as permission print strings and reference monitor callback functions. The reference monitor implements a general security policy that screens incoming requests based on the client's identity and the authentication or authorization policies it is using. The monitor does not base an authorization decision on the requested operation or the target object. The ACL manager performs that job. A default reference monitor is provided by the ACL manager. If an application has its own reference monitor, it will be invoked instead of the default monitor supplied with the ACL manager.

2. The application server must call the ACL storage manager to allow it to initialize itself. The initialization calls performed by the application server are only done once when the whole system is initialized.

3. The application manager calls the ACL manager to perform an authorization decision or to invoke a general ACL utility.

4. The application manager calls the ACL storage manager to add a new ACL to the database or to delete an old ACL from the database.

5. The ACL manager calls the ACL storage manager to transfer an ACL to or from the database in response to rdacl requests coming from acl_edit.

6. The ACL storage manager calls the ACL manager utility routines to manipulate ACL data structures. One manipulation operation involves converting permissions from human readable form into a bitmap and vice versa.

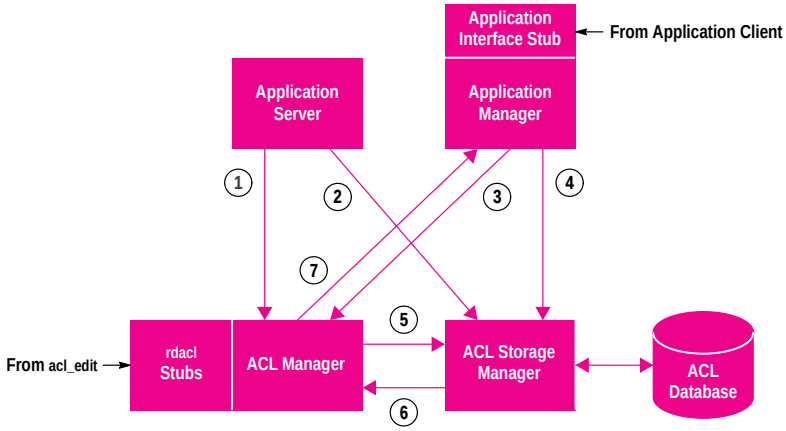

Fig. 3. Architecture for modules that make up the common ACL management module interface.

7. The ACL manager must make a callback to an application-specific reference monitor routine to screen an incoming rdacl request according to the application's general security policy.

The goal for the common ACL management module interface was to explore appropriate programmatic interfaces. Our implementation was a proof of the concept for the design and was not intended to be the best ACL manager package. The implementation provided the same functionality as the sample application except that it used an in-memory binary tree to allow applications to add ACLs at run time. The main contribution of the common ACL management module interface from an application developer's standpoint is the ability to link with a general-purpose library rather than cutting and pasting source code. The application developer can use higher-level interfaces for creating ACLs and get authorization decisions without having to understand and write the underlying mechanism.

Although the common ACL management module interface was never sold as an HP product, it was useful in several ways. First, we learned a great deal about ACL management and what developers would want to be able to do with it. Second, we used the modules in an internal DCE training class that allowed us to teach ACL management concepts and have the students add ACL management to an application they developed during a two-to-three-hour laboratory exercise. The common ACL management module interface allowed the students to spend their lab time reinforcing the concepts presented in the lecture rather than getting bogged down in writing a lot of supporting code just to make their application work. The experiences of the class reinforced our belief that it is possible to support application developers in the creation of ACL management functionality without every developer having to understand all of the complicated details of ACL management that are DCE-prescribed but not application-specific.

The version of DCE provided by OSF only supports C programmatic interfaces. It made sense to implement the common ACL management modules in C for two reasons:

- Since we were layering on top of DCE, it was more convenient to use the supported language.
- We expected that users of the common ACL management modules would also be programming in C, and so would want C interfaces to the common ACL management module interface library.

However, there is growing interest in C++ interfaces to DCE as well as support for object-oriented programming. In response to that need, a C++ class library for DCE called OODCE (object-oriented DCE) has been developed.

HP OODCE: A C++ Class Library for DCE

The common management module interface acted as a springboard for design and implementation of the C++ ACL management classes which are part of HP's OODCE product. Since it is much easier to create abstract interface definitions in C++ than in C, these DCE ACL management classes make it easier to provide access control within a DCE server. Application developers can reimplement specific classes to customize the ACL manager to fit their needs. The classes supplied with OODCE and their interrelationships are shown in Fig. 4. The classes shown in Fig. 4 represent further modularization of the ACL manager and ACL storage manager components shown in Fig. 3.

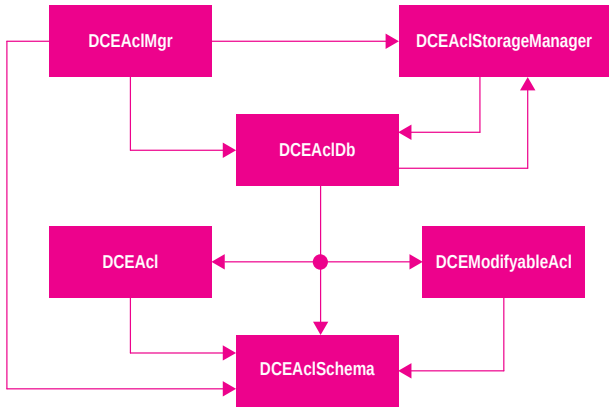

Fig. 4. HP OODCE ACL management class interrelationships.

Class Descriptions. The DCEAclMgr class implements the rdacl interface for use by the acl_edit tool and other management tools. There is one instance of an DCEAclMgr per application server. The DCEAclStorageManager manages all ACL databases for this server. The DCEAclStorageManager is responsible for finding the database in which the ACL is stored and returning a handle to that database. Programs invoke the DCEAclStorageManager interface to create or register a new ACL database and to access existing ones.

The DCEAclDB class defines the interface to an ACL database. An ACL database may define multiple 32-bit words of permissions. The interpretation of the permission bits is stored in a DCEAclSchema object, and each database has exactly one DCEAclSchema associated with it.

The DCEAcl class defines an interface for accessing DCE ACL information. In addition to the DCE ACL information, the DCEAcl class contains information about the database in which it resides, the owner and group of the protected object, and other information that is needed by an implementation. The DCEAcl's state is read-only. The DCEModifyableAcl class is a modifiable version of the DCEAcl class.

Using the OODCE ACL Management Classes. The application server invokes a simple macro that initializes the ACL system. OODCE, by default, handles all the details of making the rdacl interface ready to be invoked by remote clients. This includes registering the interface with the RPC run-time routines so that an incoming request for that interface is received and ensuring that the correct entry point for the rdacl routines is invoked. The application server also handles exporting location information to the endpoint mapper† and CDS (cell directory service) database so that clients can find the server's ACL management interface. That is the only required involvement of the application server. However, the application server may create DCEAclDb objects that can be shared across manager objects. These databases must be registered with the DCEAclStorageManager.

† The mapper maintains a list of interfaces and the corresponding port numbers where services of the interfaces are listening.

Application managers create new ACL objects by first requesting the DCEAclDb object to create a DCEModifyableAcl object and adding ACL entries to it. When done, the DCEModifyableAcl object is committed (added) to the database. To get an authorization decision, an application manager retrieves an ACL object from the database and interacts with it to get an authorization decision.

Overall, it is easier for the application developer to use the OODCE ACL manager classes than any of the previous solutions. Many of the routine tasks are done by default by the library, but they can be overridden if there are special circumstances. The ACL management objects are written to the abstract class definition so that users can provide their own implementations of DCEAclDb, DCEAcl, and DCEModifyableAcl classes and have them plug into the rest of the system.

A DCEAclDb implementation encapsulates the database access. This allows the flexibility of storing ACLs either with the objects managed by the server or in some other database. Any commercial database product can be used. The server developer need only implement DCEAclDb so that it conforms to the abstract interface and makes the calls to the commercial database of choice.

The DCEModifyableAcl class allows for fine-grained editing. The rdacl interface only supports the atomic replacement of an entire ACL, whereas the DCEModifyableAcl design supports changing individual elements within an ACL.

HP OODCE ACL objects are more general-purpose than the common ACL management module interface described earlier because the abstract class design of HP OODCE accommodates more features. Its design supports more than 32 permissions, and registration of the rdacl interface with CDS and the endpoint mapper is automatic and transparent to the server developer.

Current Status. HP OODCE is now a product. It includes default implementations for all the classes, but we expect that customers will write their own implementations of DCEAclDb and possibly of DCEAcl and DCEModifyableAcl. There is still much to learn about what distributed application developers really need from an ACL management package, but with the HP OODCE library as a product, we have more opportunity to get feedback. HP OODCE is described in more detail in the article on page 55.

Acknowledgments

I would like to thank Bob Fraley who is the codeveloper, my object-orientation mentor for the HP OODCE portion of this project, and the principal reviewer of this paper. Jeff Morgan and John Dilley were developers of HP OODCE and contributed to discussions of design and implementation of the ACL management portion. Mickey Gittler enhanced and made the HP OODCE ACL manager classes into a product. Thanks also to Jeff Morgan and Cas Caswell who reviewed this paper and gave helpful suggestions for improvement.

References

1. W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*, O'Reilly & Associates, Inc., September 1992.

2. J. Shirley, *Guide to Writing DCE Applications*, O'Reilly & Associates, Inc., June 1992.

3. *DCE Application Development Reference Manual*, Open Software Foundation, Cambridge, Massachusetts, 1991.