Approach to object security in Distributed SOM

by M. Benantar
B. Blakley
A. J. Nadalin

We briefly review the IBM System Object Model (SOM, incorporated in SOMobjects™) and Distributed SOM (DSOM). We then describe the base DSOM security architecture characterized by the presence of the Object Security Service (OSS) framework in the DSOM run-time environment. Depending on its implementation, this framework can be wrapped around procedural security service providers, thus taking advantage of existing security mechanisms. Subsequently we elaborate on the OSS elements for authentication and authorization and how they relate to the DSOM Object Request Broker (ORB™) on the one hand, and to the client and server applications on the other hand. We discuss the DSOM approach to object access control and present a novel method for enforcing it.

In a distributed environment, client and server applications cross the boundaries of a single address space and communicate with each other, passing information about service requests and receiving results. The client and the server can be running on two different hardware and software systems, remote from each other, and linked through the communication media and protocols of an underlying network. Although this paradigm of computing is effective, developing client/server applications presents some difficulties and challenges. The Open Software Foundation (OSF) has been involved in a major effort that addresses these challenges in its Distributed Computing Environment infrastructure, known as OSF DCE**,[1] or simply DCE. DCE defines the elements that make up the basic distributed computing environment, including a service that provides unique names, a security service, and a service relied on for accurate time, all within a defined network scope called a *cell*. DCE application servers advertise their network bindings and programming interfaces through the naming service, which client applications consult in order to initiate service requests. Remote operations are then triggered by a client application in a manner similar to invoking local procedures.

With the advent of object-oriented programming, the need for computing with objects over a network, transparently from the systems and the hardware platforms on which they reside, has naturally evolved. It is toward meeting this need that the Object Management Group (OMG), a consortium of computer software and hardware vendors, has developed its Common Object Request Broker Architecture**, CORBA**.[2] Its Object Request Broker** (ORB**) specification defines mechanisms through which a client is able to retrieve a reference to a remote object and invoke its methods in the same manner as when the object is local to the client application.

The IBM Distributed System Object Model provides a DSOM ORB consistent with the CORBA specification and allows inter-ORB connections in accordance

0018-8670/96/$5.00 © 1996 IBM

with the OMG ORB-interoperability specification. This allows communication and interaction with other ORB systems that conform to the OMG interoperability standard. This support greatly enhances the openness of the DSOM technology and widens the scope of its use.

While the CORBA architecture addresses the mechanisms that enable remote object interactions, the Object Management Architecture (OMA) specification defines a complete set of object services and common facilities for building distributed object solutions. These services, some of which are still being developed, include object life cycle, naming, event, persistence, and security.[3,4] In this paper we focus on the security service designed and integrated with DSOM and developed in accordance with the OMG security requirements and guidelines.[4,5] In the next section we introduce the fundamentals of SOM and DSOM. In following sections we: describe the DSOM security architecture and show how it relates to the DSOM ORB kernel; discuss the establishment of a secure association between DSOM client and server entities; elaborate on the object framework defining the DSOM security architecture; present the DSOM approach to object access control and its enforcement; and show how end-user client and server application developers can interact with the DSOM security framework.

SOM and DSOM overview

SOM (SOMobjects*) provides the key object-oriented programming features, including inheritance and polymorphism.[6] In addition, SOM makes use of the *metaclass* concept that extends the object definition to the class itself—a SOM class is itself an object that responds to the interfaces or methods defined by its metaclass. SOM provides for software development with objects based on a class inheritance hierarchy. At the root are two basic SOM run-time classes, SOMObject and SOMClass. The former is the root ancestor for all classes defined in a SOM application and the latter is the root ancestor of all SOM metaclasses. A SOM object is an instance of a SOM class, and all SOM classes defined by an application are ultimately derived from SOMObject, including SOMClass. Similarly, all metaclasses of SOM classes are ultimately derived from SOMClass. Because they are classes, both SOMObject and SOMClass are instances of the root metaclass, SOMClass. The SOMObject class introduces generic methods that, by inheritance, are applicable to all SOM objects of an application. These methods provide basic SOM functionality, such as the dispatch mechanism for executing method invocations. Similarly, all SOM class objects respond to the generic methods defined by SOMClass, including a method for creating instances of a class, and methods for creating and modifying instance methods of a class.[7] A SOM application program can be organized as a set of directed acyclic graphs of classes and subclasses, where computation consists of invoking methods on objects instantiated from this class hierarchy. Figure 1 shows the object and class relationships found in a SOM application.

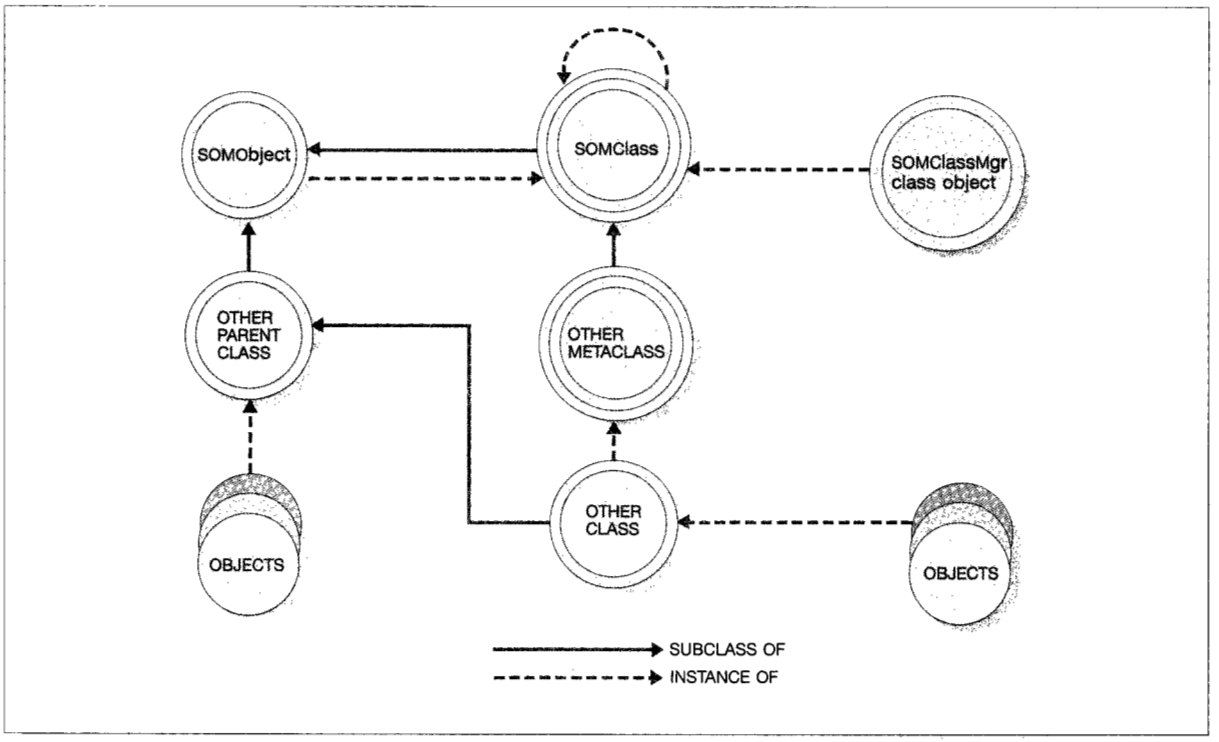
The security service integrated with DSOM was developed in accordance with OMG requirements and guidelines.

Distributed SOM, or DSOM, adds the functionality of an ORB to SOM. A DSOM client application seamlessly creates and invokes methods on remote objects through the run-time mechanism of *proxy* objects. When a remote object is created through the DSOM ORB, a reference to a local proxy object is returned to the client. This object then becomes the local representative of the remote object. It receives service requests through method invocations and forwards them to its corresponding real object. At the time the proxy object is created, the interface to the remote object is obtained from the *Interface* Repository database maintained by the SOM compiler.

The *Implementation* Repository, on the other hand, is used to retrieve information about the implementation of the server. This includes its host, its implementation identifier, the class of its server object, and the path for loading its executable code. The server-side run-time environment includes a number of objects. Most important are the SOM Object Adapter (SOMOA) and the application-specific server object, which is generally an instance of the predefined SOMDServer class or one that is user-defined and derived from SOMDServer.

The SOMOA object defines the main interface between the server program and the DSOM run-time environment. After a connection to the server has

Figure 1 Class hierarchy of a SOM/DSOM application



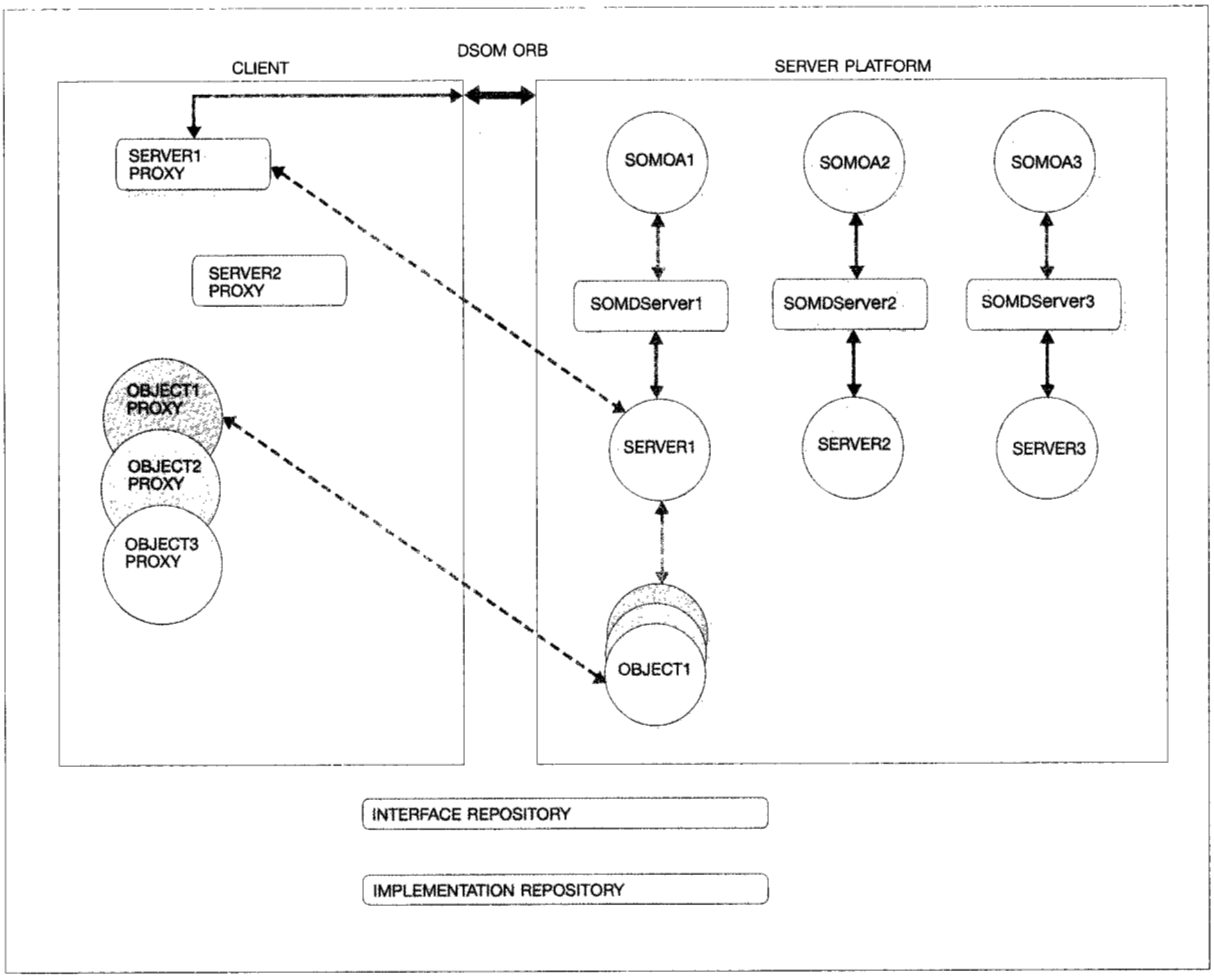
been established, the SOMOA object receives client requests for services, and in cooperation with the SOMDServer object it creates, resolves DSOM references to local objects and dispatches methods on these objects. In addition to providing an interface to the SOMOA object, the server object allows object creation and destruction by client applications and provides object management services needed by clients. Figure 2 depicts the basic elements involved in DSOM client/server interactions.

DSOM security architecture

The functionality of DSOM security is encapsulated by the Object Security Service (OSS) framework. The OSS component is transparent to end users; however, security-aware applications can use the OSS interfaces to alter run-time security according to their unique requirements. The local ORB then applies the selected security features to secure the communication of the local entity with a remote entity. The encapsulation provided by OSS isolates the security functions from the rest of the DSOM kernel. It provides flexibility to implementors concerned with having clear boundaries between a trusted computing base and the rest of the system in their particular platform, e.g., the MVS (Multiple Virtual Storage) operating system. In addition it provides a method for the DSOM security implementation to be independent of underlying mechanisms, as OSS itself can bridge to different security mechanisms or security service providers (SSPs).

Distributed computing with objects is a client/server paradigm in which a client application locates a remote object reference, binds to its serving application, and invokes methods on the target object. From the security perspective we must prove the identity of the client to the serving DSOM application and also provide the ability for the serving application to authenticate itself to the client. This establishes a peer-to-peer level of trust and achieves a mutual authentication "handshake." Once this is accomplished, subsequent communication between the client and the server entities carries the seal of the established security context. Optional access control to the server's objects will also be enforced by OSS. It is at the discretion of a server application whether or not to

Figure 2 Basic DSOM client/server interactions



protect its objects. Since authorization functionality is in the OSS framework, rather than the DSOM kernel, no overhead is incurred when authorization is not used. As illustrated in Figure 3, the DSOM ORB engages in a two-way interaction, requesting security services from OSS and receiving responses, while end-user client and server applications affect the settings of the OSS environment in a one-way interaction.

Establishing a secure client/server association

To establish a connection with a server in a separate address space or on a remote host machine, the DSOM ORB for the secure client transparently communicates the client's *credentials* and *privileges* to the server's ORB. A credential is something that can be used to prove the client's identity—for example, a secret password or a Kerberos ticket.[8] While a privilege represents a special authority, it might simply be a list of groups of which the client is an authorized member. The authority of each group in the list is then granted to the client. Client privileges are used by the server in enforcing specific access control policies for its objects. The process of communicating a client's credentials and privileges to a server is referred to as establishing a *shared security context*. This process is transparently performed through the cooperation of the client and the server

Figure 3 DSOM security architecture layers

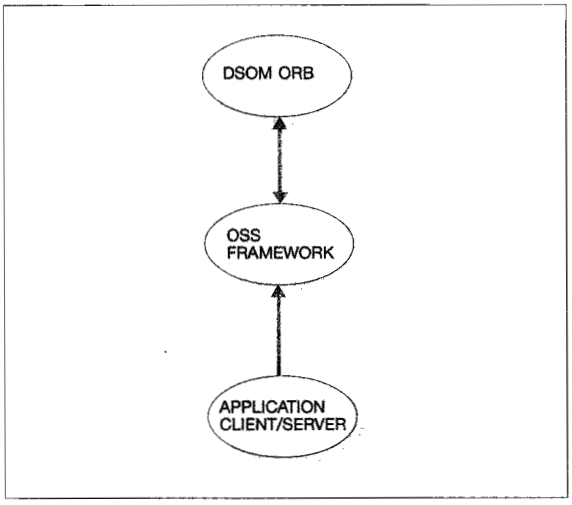
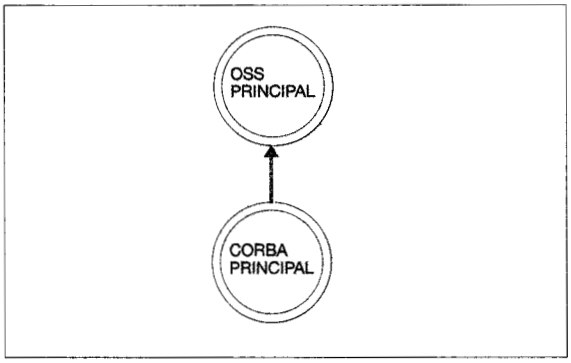


Figure 4 The CORBA Principal class inherits from the OSS Principal class



ORBs, and results in the creation of an OSS *principal* object, local to the secure server and representing the secure client, that the server uses to compute access decisions to its protected objects. The principal object holds the client's credentials and privileges.

The CORBA specification requires that an instance method, called get_principal, be defined by the object adapter class or its subclass (SOMOA for DSOM) to allow the server application to retrieve the identity (ID) of its caller at any point during its application. SOM implements this method by returning an object instantiated from its CORBA-compliant Principal class. This principal object can be queried, using the get_userName method, to obtain the ID of the client.

We redefine the Principal class to be a subclass of the OSS Principal class. Thus invocation of the get_principal method returns an object, with all of the CORBA-compliant behavior expected by SOM, that also encapsulates a set of privileges defining the scope of the client's authority while engaged in service requests.

In a prototype integration with DCE security, the client privileges were represented by a DCE privilege attribute certificate (PAC) that the DCE access control mechanism can interpret. Similarly, these can be in a format useful to any other supported access control mechanism. Figure 4 shows the relationship between the OSS and the CORBA principal objects.

Authentication objects and flows

At run time, OSS contains a set of objects—some that encapsulate authentication functions and others that deal with object access control. These two functions are, generally, tightly coupled with each other. The shared security context established upon successful authentication of a principal (user or application) is the foundation for controlling access to a server's objects.

There are four primary authentication objects: the quality of protection, the persona, the shared security context, and the vault objects.

The quality of protection object. The quality of protection (QOP) object encapsulates security services used by the ORB run-time environment. Security-aware applications also may use the QOP interface to query and set QOP information for their particular security needs. The local ORB uses the information encapsulated by the QOP to set up the user's or the application server's local security constructs as well as the shared security contexts. Among the data present in a QOP is an indicator of the desired level of message protection during a DSOM client/server exchange. In turn, this includes different levels of message integrity and privacy (encryption), and possibly a particular integrity or message privacy algorithm.

In addition to message protection, the QOP object indicates the authentication protocol that a client will use in authenticating itself to the server, or that a

server requires for such authentication to take place. This can be either a third-party protocol or a two-party protocol. In the former the authentication token is acquired by the client from a third-party network security server, and in the latter the token is acquired from the local operating system.

A client may indicate in its QOP object whether or not its local ORB should require a remote server to authenticate itself to the client's ORB prior to engaging in service requests. Current prototype implementation provides for either the local operating system authentication token in the form of a password, a DCE authentication token, or a token generated by the IBM NetSP Secured Logon Coordinator (NetSP SLC).[9] This is a third-party network program used to authenticate the identities of two communicating network principals to one another, whether users, applications, or a combination of both.

The flexibility of DSOM authentication results from adopting the Generic Security Services Application Programming Interface (GSS-API),[10] a standard interface adopted by X/Open Ltd., integrated by OSF in its DCE 1.1 release and by IBM in its NetSP SLC product. The philosophy behind this standard is to isolate the communication protocol from the underlying implementation of the security mechanism. In our case, it is the DSOM ORB that becomes independent of the security mechanisms by using GSS-API. Although the services provided by GSS-API are transparent to the requesting applications, NetSP SLC provides advantages over the DCE implementation. The technology used by NetSP is optimized for network-layer applications due to its small token size, flexible connectivity, and access to the network security server. Either a client or a server has the ability to initiate an authentication session. In addition, NetSP has a "lighter" run-time environment than DCE.

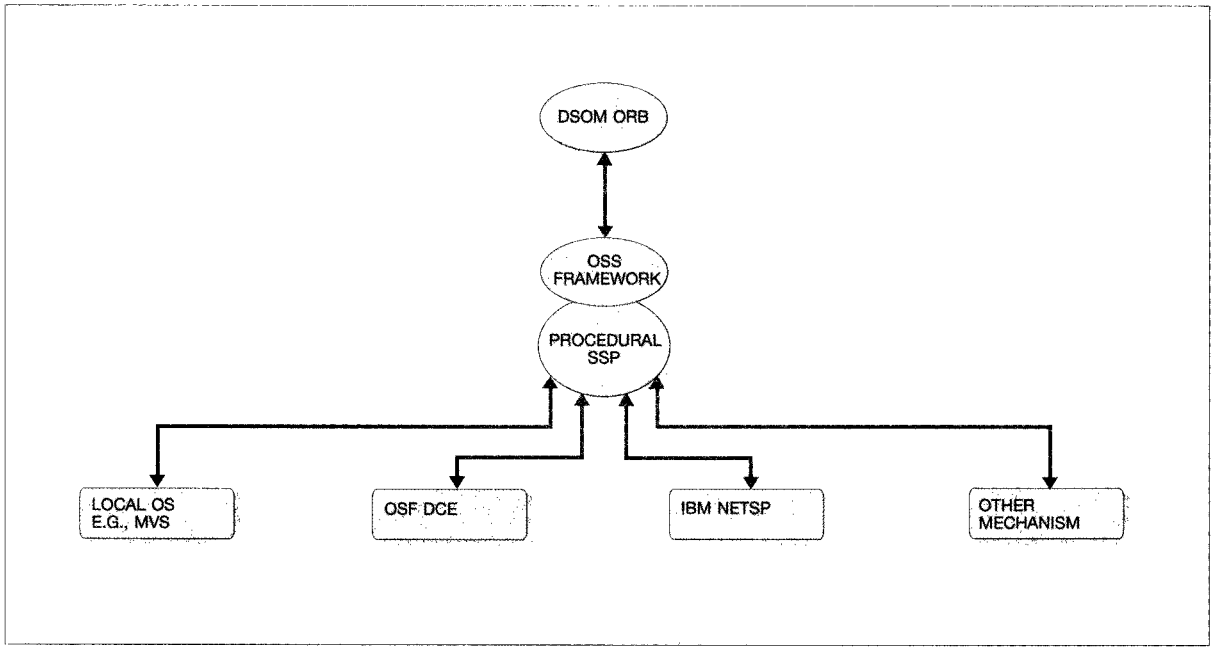
The persona object. The persona object represents an authenticated identity to the local ORB and defines an established local security context that encapsulates the credentials and privileges of its identity. Without a valid persona, a client's request for remote services is rejected by the local ORB. Similarly, without a persona a server application is prevented by its local ORB from receiving connections from its clients.

The shared security context object. While a persona relates an identity to its local process or address space, the shared security context object is created only in conjunction with a DSOM client/server dialog; thus, it is a relation between two separate and possibly remote address spaces. This relationship defines the type and the parameters of the secure association that are applicable to the communication exchanges between a client and a server. It includes a common QOP object supported by both the ORB at the client and the ORB at the server, an authentication token proving the identity of the client, and the client's privileges as defined and interpreted by the security service provider indicated in the QOP object. Upon successful creation of the shared security context object by the client and its acceptance by the target server, authentication is established and the security parameters driving the communication between the two entities become active. Usually the client ORB sets up those parameters, and it is up to the server's ORB to refuse or accept them according to the security services that it supports.

The vault object. In order to isolate the DSOM authentication services and establish a trust boundary, the authentication objects just described reside within the scope of the vault object, thus hiding and encapsulating the security constructs. The QOP, the persona, and the shared security context objects are manipulated only through methods defined on the vault object. Outside the vault object, persona and shared security context objects are referenced through "opaque handles" that protect their encapsulation. In our prototype implementation, the client and the server run-time ORBs invoke DSOM methods on the vault object, which is part of the OSS framework, while the vault object communicates with underlying security service providers through a GSS-API layer (Figure 5). This implementation of OSS was built by the Object Services Technology Center on the OS/2* (Operating System/2*) Warp and AIX* (Advanced Interactive Executive*) 4.0 platforms in order to validate the model. Experiments were conducted using different types of authentication protocols, including a two-party protocol based on local operating system registries and third-party protocols that include IBM DCE 2.1 and IBM NetSP.

Authorization is encapsulated through the system authorization policy (SAP) and the system authorization oracle (SAO) objects in the server's run-time environment. The SAP object is responsible for maintaining and retrieving the security attributes on protected objects; the SAO object is responsible for access control decisions. Both of these objects are architected so that they can become "wrappers" around existing procedural authorization mechanisms.

Figure 5 OSS flows using a procedural SSP



DSOM approach to object access control

The access control mechanism in an object-oriented system provides natural protection—the data encapsulated by an object can be transformed only through its method interface.[11,12] Object access control may fit with the evolving role-based access control, or RBAC.[13,14] In the RBAC model, access to resources is based on the transformation that would be effected on the resource as a result of the access. Thus, the transformation is confined to the initiator's role in his or her organization. In this respect, access control in object systems presents a departure from its counterpart in procedural systems.

Traditionally, control of access to a resource has been based on whether or not the data can be disclosed, altered, or destroyed. Controlling access to objects at the interface level implies that the space of access rights, or the set of allowable permissions, becomes proportional to the number of methods in the class library. We need to determine a fixed set of usable and manageable access rights that can be used by application developers to provide secure class libraries. Having a fixed set of rights is important for application portability and for reducing the complexity of access control administration.[15]

We address this problem by breaking an access right into two components: a type and a value. Our goal is to define a small number of families of access rights, each containing a manageable set of permissions. A family, or access-right type, is intended to be generic and to encompass a wide semantic scope, while an access-right value is more specific within its type. An access-right type can be thought of as being equivalent to the POSIX (Portable Operating System Interface for Computer Environments) semantic[1] of a DCE Access Control List manager. The example in Figure 6 shows two access-right types, OPERATION_RIGHTS and ROLE_RIGHTS.

Instead of using the DSOM kernel to intercept a method request in order to check access control, we encapsulate this function into OSS by making use of the SOMMBeforeAfter metaclass framework. A class that is directly or indirectly an instance of the SOMMBeforeAfter metaclass inherits "before" and "after" behavior. This means that the execution of any method on an instance of the class will be preceded by the execution of a before method and followed by the execution of an after method. The SOMMBeforeAfter metaclass implementation overrides the method somDispatch introduced by the SOMObject class. As a result, stubs are placed in

the class's method table[16] to call the somDispatch override. A stub looks like this:

```
somDispatch( self, primaryMethod, . . . ) {
    retval = BeforeMethod( class( self ),
                           self,
                           primaryMethod, . . . );
    if ( retval ) {
            primaryMethod ( self, . . . );
            AfterMethod( class ( self ),
                           self,
                           primaryMethod, . . . );
    }
}
```

where primaryMethod is the target method to be executed on an object of the underlying SOM class. The SOMMBeforeAfter metaclass provides an interception point from which access control can be triggered. Thus, by overriding the before method to perform authorization checking, the requested method will be executed only if the client is authorized for the object of the protected target class and method.[17]

Figure 6 SOM OSS access-right types

Access-Right Types	Access-Right Values	Intended Interpretation
OPERATION_RIGHTS	R	Read
	W	Write
	X	eXecute
	C	Create
	D	Delete
	A	Append
ROLE_RIGHTS	G	Guest
	U	User
	O	Operator
	M	adMinistrator
	T	audiTor
	S	Super

Figure 7 Protected objects inherit from the DSOM secure class

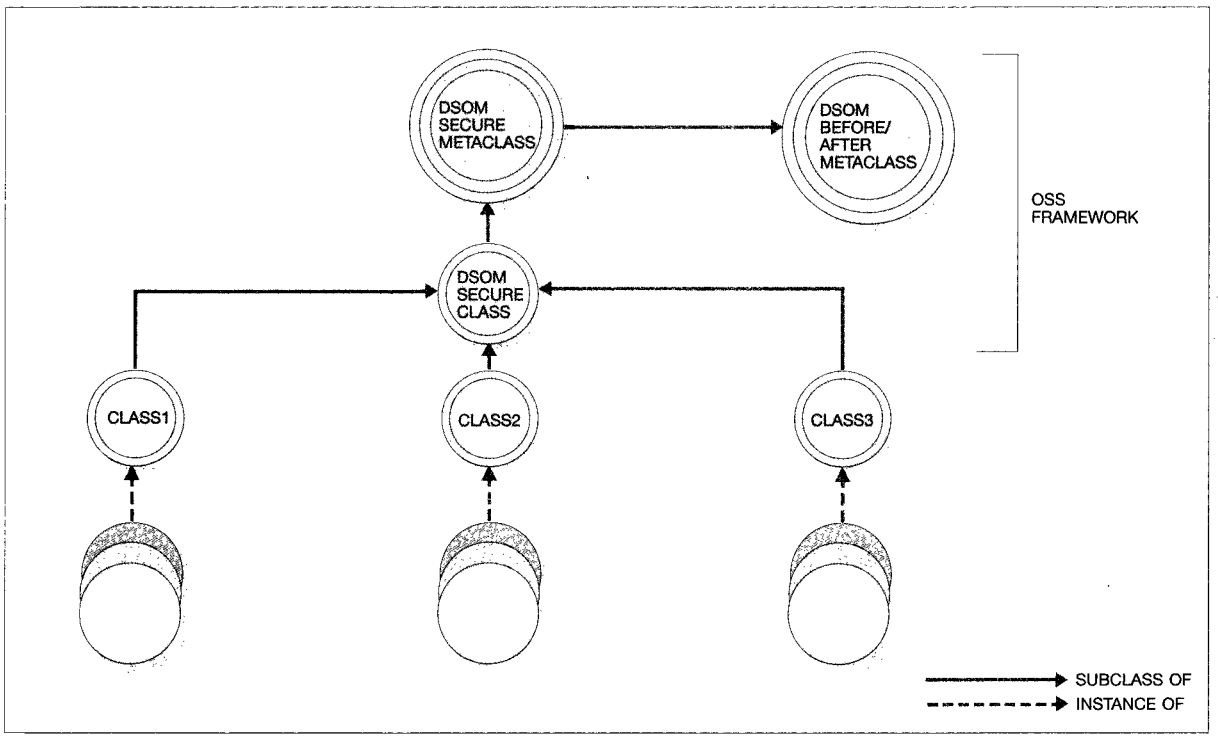
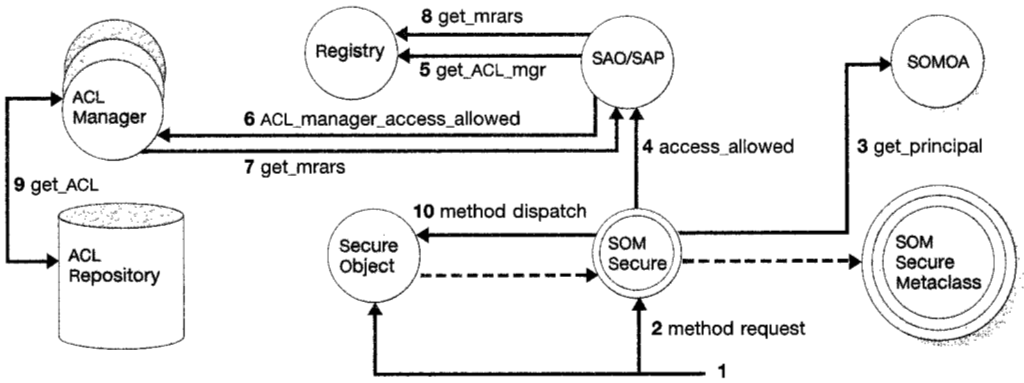


Figure 8 SOM OSS access control flows



Step 1: A method request arrives against a secure object (an instance of SOMSecure).
Step 2: The method request goes to SOMSecure (an instance of SOMSecureMetaclass).
Step 3: SOMSecure's before method issues the get_principal method against the SOMOA object to obtain the requestor's ID.
Step 4: SOMSecure's before method queries the SAO/SAP objects, using the access_allowed method (SAO=system authorization oracle; SAP=system authorization policy).
Step 5: The SAP sends a get_ACL_mgr request to the Registry object (ACL=access control list); the Registry returns the list of ACL Managers that control access to the secure object.
Steps 6-9 are repeated for each ACL Manager in the list.
Step 6: The SAP sends an ACL_manager_access_allowed request to the ACL Manager.
Step 7: The ACL Manager object asks the SAP to retrieve the mrars, via the get_mrars method (mrars=method required access rights).
Step 8: The SAP asks the Registry for the mrars, via the get_mrars method (if they are not already in the cache), and returns them to the ACL Manager.
Step 9: The ACL Manager gets the ACL entry from the ACL Repository database (if it is not already in the cache). The ACL Manager evaluates all the access rights required by the method against those held by the requestor.

The SAO reviews all the decisions of the ACL Managers and makes the final decision on whether or not the method can be dispatched.

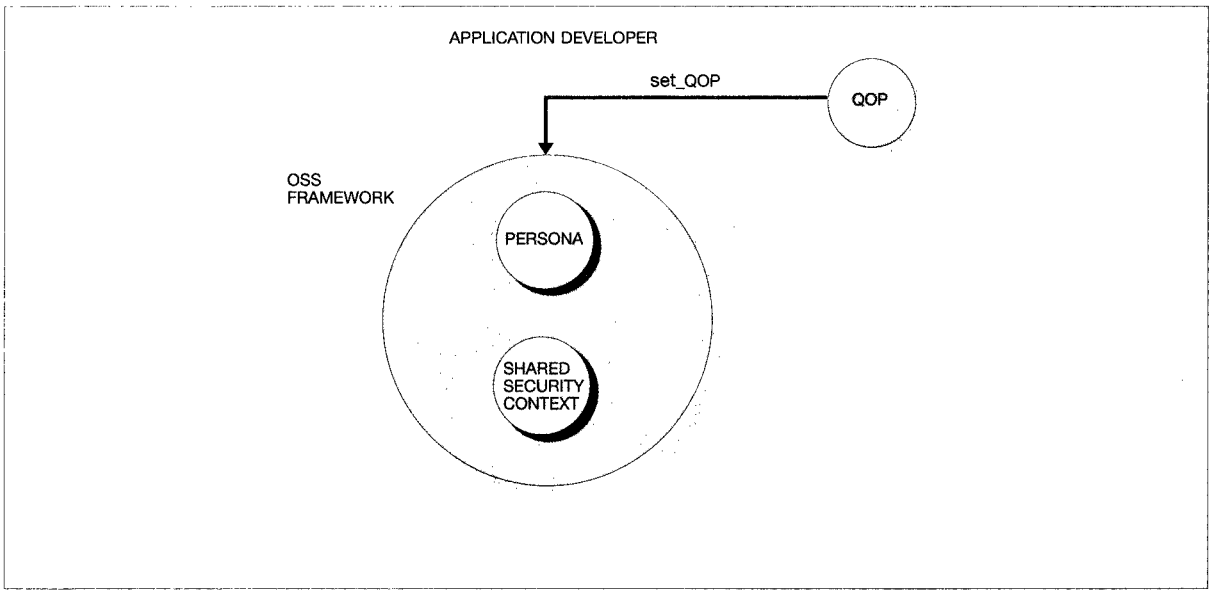
Step 10: If the SAO approves, the SOMSecure before method dispatches the requested method against the secure object.

The strength of this access control technique is characterized in two ways:

- The before behavior during which access control takes place cannot be executed on the target object's proxy that resides on the client side; it executes only on the real object within the server. Thus, the client is prevented from tampering with the authorization process.
- If a client application attempts to directly invoke somdDispatchMethod[6] on the server's proxy, SOMDServer, or a descendant of it, then the request automatically fails.

In this way, DSOM access control is seamlessly enabled by making a class, designated to be protected by its developer, inherit from the DSOM secure class provided by the OSS framework as shown in Figure 7. Our prototype included an implementation of a native capability model as well as an integration with the access control list (ACL) mechanism based on IBM DCE 2.1. Figure 8 shows the execution flow that com-

Figure 9 A DSOM client or server setting up a new QOP object



putes an access decision (step 4) prior to dispatching a method on a secure object (step 10). Although current SOMobjects 3.0 makes use of a native DSOM mechanism as a Security Service Provider, later releases of SOMobjects will integrate with DCE for both authentication and authorization services.

Application developer interaction with OSS

Developers of client or server applications may not want to use the default ORB security features, but instead customize the security based on their own requirements. If the selected security features are available through OSS, a DSOM principal can be developed that creates a new QOP object, sets its instance values, then communicates with the OSS environment using the set_QOP method on the vault object, as shown in Figure 9. Subsequent shared security contexts will be created under the new QOP object.

When a server application is started, it usually acquires the security context of the principal, possibly an end-user administrator who started it. In order for a server to engage in a complete mutual authentication process with its clients, it should run under its own identity, and hence its own security context. A server application, once started, can run under its own identity by creating its own persona object using the instantiate_persona method on the vault object, then making the new persona object active in the OSS environment by invoking activate_persona, as illustrated in Figure 10. Note that running a server under its own authenticated identity encapsulates the trust into a program principal, rather than a human principal who may be susceptible to voluntarily compromising this trust.

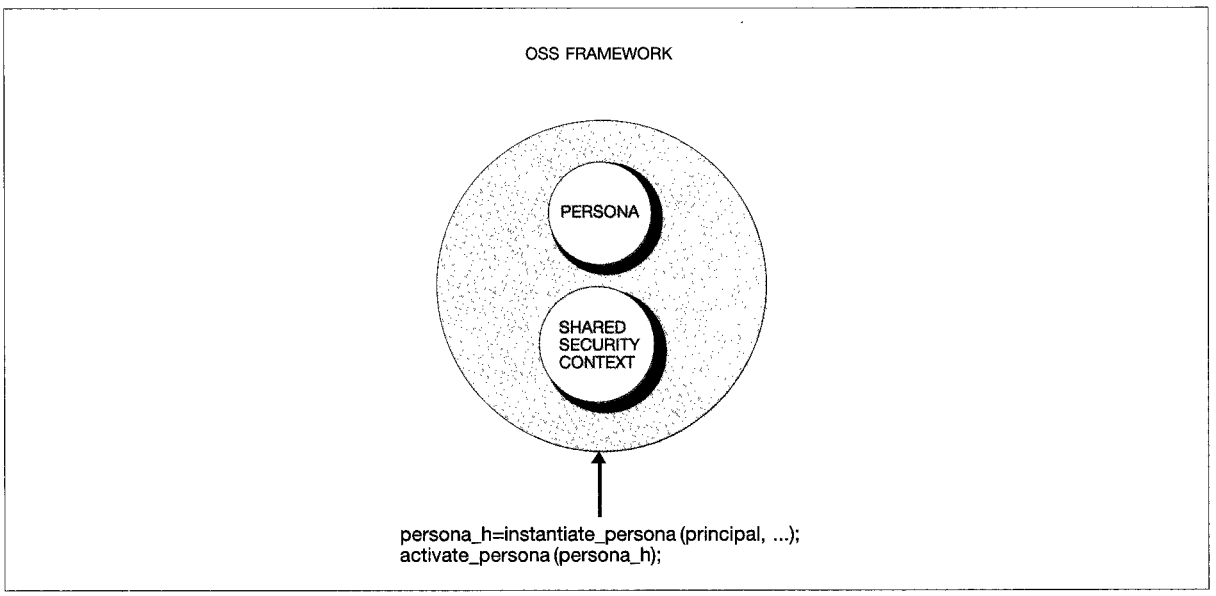
Conclusion

We have designed an object security framework that is an integral part of DSOM object services and defined its boundaries with respect to the DSOM kernel ORB and to application developers. This solution is characterized by two factors:

- It is seamless to end-user clients and application developers in that it is the duty of the DSOM object request broker to transparently provide security.
- The solution is designed to be independent of underlying security services. Furthermore, it can make use of the existing procedural security service providers such as the DCE security services.

We described an authorization model based on method-level granularity. We introduced its basic el-

Figure 10 A DSOM application server changing to a new identity



ements, including a way to manage the set of allowable access rights. We presented a novel way to enable DSOM object access control that simply makes use of the existing SOM before/after metaclass framework. This method is safe from client tampering and, although unique to DSOM, it can be adopted by other object systems providing support for the before and after behavior or its equivalent.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of the Open Software Foundation or the Object Management Group.

Cited references

1. Open Software Foundation, *OSF DCE Application Development Reference*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1993).
2. *The Common Object Request Broker: Architecture and Specification, Revision 2.0*, Object Management Group, Framingham, MA (1995).
3. *White Paper on Security, Document 94.4.16*, Object Management Group, Framingham, MA (1994).
4. *Object Services Request for Proposals (RFP3)*, Object Management Group, Framingham, MA (1994).
5. S. L. Chapin, W. R. Herndon, L. Notariacomo, M. L. Katz, and T. J. Mowbray, "Security for the Common Object Request Broker Architecture (CORBA)," *Proceedings of the 10th Annual Computer Security Applications Conference*, Orlando, FL (December 1994), pp. 21–30.
6. *SOMobjects Users Guide*, SC23-2680, IBM Corporation (1994); available through IBM branch offices.
7. *SOMobjects Program Reference*, SC23-2681, IBM Corporation (1994); available through IBM branch offices.
8. M. Benantar, R. Guski, and K. M. Troidle, "Access Control Systems: From Host-Centric to Network-Centric Computing," *IBM Systems Journal* **35,** No. 1, 94–112 (1996).
9. *Network Security Program Security Developer's Guide*, SC31-6500-01, IBM Corporation (1994); available through IBM branch offices.
10. J. Linn, "Generic Security Service Application Program Interface," *Internet Draft* (June 1991).
11. C. Bryce, "Protection in Object-Oriented Software," *Proceedings of the 2nd International Conference on Achieving Quality in Software*, Venice, Italy (October 1993), pp. 97–109.
12. P. S. Deng, "Fast Control in Object-Oriented Repetitive Access," *IEEE Proceedings of the 28th Annual International Carnahan Conference on Security Technology*, Albuquerque, NM (October 1994), pp. 173–175.
13. D. Ferraiolo and R. Kuhn, "Role-Based Access Controls," *Proceedings of the 15th National Computer Security Conference*, Baltimore, MD (October 1992).
14. R. S. Sandhu and H. Feinstein, "A Three Tier Architecture for Role-Based Access Control," *Proceedings of the 17th National Computer Security Conference*, Baltimore, MD (October 1994).
15. G. Karjoth, "A Formal Model of OSS Authorization," unpublished paper, available on request from gka@ibm.vnet.com.
16. I. Forman, S. Danforth, and H. Madduri, "Composition of Before/After Metaclass in SOM," *OOPSLA '94 Conference Proceedings*, Portland, OR, ICM Press (1994), pp. 427–439.
17 M. Benantar, B. Blakley, and A. Nadalin, "Use of DSOM Before/After Metaclass for Enabling Object Access Control," presented at the *IFIP/IEEE International Conference on Distributed Platforms (ICDP96)*, Dresden, Germany (February 27–March 3, 1996).

Accepted for publication January 22, 1996.

Messaoud Benantar *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (electronic mail: benantar @vnet.ibm.com).* Dr. Benantar is an advisory programmer with the RACF (Resource Access Control Facility)/MVS development group in Poughkeepsie. He received his diploma d'ingenieur in computer science from the Université des Sciences et de Technologie, Algiers, in 1983, and the M.S. degree in 1986 and the Ph.D. degree in 1992, both in computer science from Rensselaer Polytechnic Institute. He spent one year on assignment at the Object Services Technology Center in Austin, Texas, as a member of the IBM interdivisional team working on the object security services project. His interests include system and network security, object-oriented computing systems, and parallel algorithms.

Bob Blakley *IBM Personal Systems Products Division, 11400 Burnet Road, Austin, Texas 78758 (electronic mail: blakley @vnet.ibm.com).* Mr. Blakley, who joined IBM in 1989, is the OS/2 LAN system security architect. He is also the lead security architect for the Personal Systems Products (PSP) Division and in this role is responsible for the development of security technology to be included in the SOM product line. He is IBM's security representative to the OMG and coedited the OMG CORBA security standard, which defines the interface and services needed to secure a CORBA-based object-oriented environment. He is the OS/2 LAN Systems representative to the IBM Security Architecture Board and the LAN Systems representative to the OSF Security Special Interest Group. He has served for two years as the chair of the OSF Distributed Management Environment/ Distributed Computing Environment security working group. Mr. Blakley holds an A.B. degree in classics from Princeton University, and M.S. and Ph.D. degrees in computer and communications sciences from the University of Michigan. He has been involved in cryptography and data security design work since 1979.

Anthony J. Nadalin *IBM Personal Systems Products Division, 11400 Burnet Road, Austin, Texas 78758 (electronic mail: anthonyn@vnet.ibm.com).* Mr. Nadalin joined the IBM Federal Systems Division in 1983, where he worked on secure projects for the government. In 1989 he began working on secure operating systems design. This work, including evaluations of the MVS and VM (virtual machine) operating systems, was in support of the IBM development laboratories, with the goal of developing commercial "off-the-shelf" secure operating systems. In 1992 he transferred to the Application System/400 Division to complete an evaluation of secure operating systems and databases. While on special assignment to the PSP division he worked on the specification and prototype of OSS. In 1995 Mr. Nadalin joined the PSP division, where he is now part of the security design team for base operating system and distributed computing and is assisting in the transfer of the OSS prototype to the SOMobjects group.

Reprint Order No. G321-5601.