Analysis of Authorization in CORBA Security

Günter Karjoth
IBM Research Division, Zurich Research Laboratory
gka@zurich.ibm.com

December 3, 1996

Abstract

Integration of security and object-oriented techniques is critical for the successful deployment of distributed object systems. In December of 1995, the Object Management Group published a security service specification, called CORBA Security, for handling security in object systems conforming to the Object Management Architecture. This paper provides a rigorous definition of the authorization part of CORBA Security. Its semantics is given in terms of an access control matrix. The dependencies among the authorization elements are analyzed and possible interpretations for access control decision functions are given.

Keywords: authorization, object access control, CORBA, security, distributed object systems

1 Introduction

The Object Management Group (OMG), a 500 member strong consortium, has developed an object model and a reference architecture for the construction of applications interworking across diverse architectures and infrastructures using object-oriented techniques. The Object Management Architecture [8] defines the necessary facilities including at its core the Object Request Broker (ORB), a mechanism that provides transparency of object location, activation, and communication. The Common Object Request Broker Architecture (CORBA) specification describes the interfaces and services that must be provided by compliant ORBs. Object Services, to be shared by applications across all domains, provide the basic functions for realizing distributed object solutions. Naming, events, persistence, transaction, and concurrency control are examples of object services. Security is regarded as a service as well. In their Object Services RFP3 [7], including the OMG White Paper on Security [6] as an annex, OMG defines the requirements for security in object systems conforming to the Object Management Architecture.

In December of 1995, OMG issued the CORBA Security (CS) specification [10], which defines the core security facilities and interfaces required to ensure a reasonable level of security of a CORBA-compliant system as a whole. The CS specification includes a security model and architecture, the security facilities and interfaces available to application developers, administrators, and object system implementors, and a description of how security services affect the CORBA2 interoperability protocol [9]. The reference model describes how and where a secure system enforces security policies. The security architecture implements the security model. As security services have implications on the ORB and on most Object Services, Common Facilities, and object implementations, the CS specification also includes rules to express conformance to CORBA Security.

CS provides services for authentication, delegation, authorization, quality of protection including privacy, non-repudiation, and audit. This visible security functionality uses other security functionality such as cryptography, which supports many of the above functions but is not visible outside the security services. Thus, to achieve independence of security technology, CORBA Security does not define cryptographic interfaces, nor does it propose any direct use of cryptography by application objects.

Typically, distributed object systems support many more objects, generally several orders of magnitude more, than language-based object systems [5]. For scalability, CS authorization provides the notions of privilege attributes (user groups, roles, etc.) and object groups to reduce the cardinality of the sets of subjects and objects and thus the size of the access matrix. Methods of an object's interface can also be grouped by assigning appropriate rights required for their invocation. This assignment partitions an object interface with different users/groups being authorized to different partitions.

CORBA Security treats authorization as an independent semantic concept that is separated from its implementation in system-specific mechanisms. It is therefore necessary that the language representing the authorization must have a formal semantics so that the meaning of an authorization statement can be precisely determined. This way, the security administrator as well as the CS implementor have the same understanding. This is particularly important as vendors of CORBA products begin to include implementations of the security specification, e.g. the security service integrated with DSOM (IBM's Distributed System Object Model) was developed in accordance with OMG requirements and guidelines [2].

This paper provides a formal definition of authorization in CS as defined in document [10]. The semantics is given in terms of a protection state, the set of rights held by each individual subject, and an authorization scheme; i.e. a set of rules that lets the protection state evolve by the autonomous activities of the subjects. The dependencies among the authorization elements are analyzed and possible interpretations for access control decision functions are given. Section 2 provides a brief description of CS authorization. Section 3 introduces the formal model on which CS authorization becomes defined. The specification of CS authorization is developed in Section 4. It describes the elements of CS authorization, defines possible access decision functions, and elaborates on the semantics of operations in the CS administration interface. Section 5 presents a mandatory access control policy expressed within the CS model. Finally, Section 6 contains a discussion of open issues and presents conclusions.

2 Access Control in CORBA Security

CORBA Security manages security policies based on the security attributes of resources and principals. It provides abstractions to reduce the size of access control information but, on the other hand, allows fine-grained access to individual operations rather than to the object as a whole. CORBA Security interfaces are object-oriented and shall allow polymorphic implementations of its constituent objects based on different underlying mechanisms. [10] defines a particular access policy type based on privilege and control attributes and an associated management interface. All access decisions on object invocation are made via a standard access decision interface, no matter which objects define the policy and implement the access decision.

Principals are users or processes accountable for the actions associated with some user. Depending on a given security policy, a principal may have certain privilege attributes which can be used for access control such as access identities, roles, groups, security clearance, or capabilities. At any time, the principal may choose to use only a subset of the privilege attributes it is permitted to use in order to establish its rights to access objects. On the other hand, control attributes are associated with the target. Examples are ACLs and information used in label-based schemes.

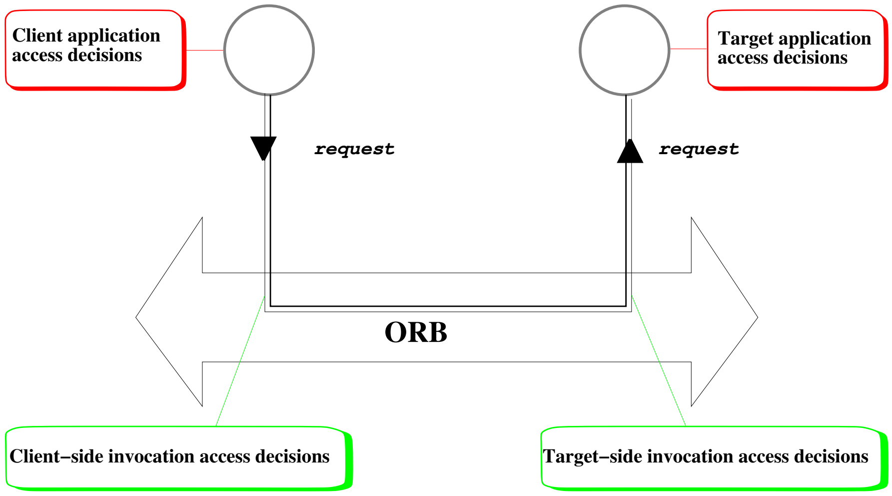

Figure 1: The Access Control Model

The access control model, depicted in Fig. 1, allows access control policies to be defined at two levels: at object invocation and within the application. The access decision function bases its decision on the current privilege attributes of the principal, the operation to be performed, and the control attributes of the target object. Although applications can enforce their own access policy, no standard application access policy is defined. Therefore, we deal only with object invocation access policies and focus on access policy controlled at the target side, the most interesting case.

A *Security Policy Domain* is the set of objects to which common security policies apply. The domain itself is not an object. Security policy domains permit application of security policy information to security-unaware objects without requiring changes to their interfaces (by associating the security policy management interfaces with the domain rather than with the objects to which policy is applied). Security policy domains provide leverage for dealing with the problem of scale in security policy management (domain granularity rather than object instance granularity).

The CS reference model allows objects (and domains) to be members of multiple domains. The policies that apply to an object are those of all its enclosing domains. However, document [10] does not specify policy composition rules; problems of consistency, complexity, and administration are not addressed either.

Privilege Attribute	Delegation State	Granted Rights
access_id:a_1	initiator	`corba: gs-` `other: -u-m-s`
access_id:a_1	delegate	`corba: g--` `other: ------`
group:g_1	initiator	`corba: g--` `other: -u---s`
group:g_2	initiator	`corba: gs-` `other: ------`

Figure 2: Domain Access Policy (Granted Rights)

A *Domain Access Policy* grants a set of subjects a specified set of rights to perform operations on all objects in the domain. A representation of a particular Access Policy is given in Fig. 2. This table resembles the traditional access control list and defines the rights a principal has on objects in a certain domain.

Compared with the structure of a usual access control list we notice three differences. First, to simplify administration, a Domain Access Policy aggregates principals for access control by using their privilege attributes as subject entries. User identities are considered to be a special case of privilege attributes. Second, a Domain Access Policy allows to grant different rights when a privilege attribute is used by a delegate than is used by an initiator. Third, to cope with different needs to express authorization, rights are qualified into sets of "access control types", named *rights families*. However, CS defines only one RightsFamily[1], the "corba" family, containing the three rights `g` (get), `s` (set), and `m` (manage). Rights are always qualified by the name of the RightsFamily to which they belong. CS implementations are free to define and provide additional rights families although doing so is discouraged to

[1]Objects/classes occurring in [10] are denoted by capitalized names.

keep the complexity of the administrator's model low.

Required Rights	Rights Combinator	Operation	Interface
`corba: -s-`	all	m_1	c_1
`corba: gs-`	any	m_2	
`corba: gs-`	all	m_1	c_2
`other: ---m-s`	all	m_2	
`other: g-om-s`	any	m_1	c_3

Figure 3: Required Rights

An AccessPolicy object grants rights to privilege attributes. In contrast, a RequiredRights object defines that for the invocation of each operation in the interface of a secure object some set of rights is required. An example of a RequiredRights object is shown in Fig. 3 in the form of a table. This table defines the rights required to gain access to each specific method of an object. There is also a mechanism to specify whether a user needs *all* the rights in a method's required rights entry to execute that method (AND semantics) or whether it is sufficient to match *any* right within the entry (OR semantics).

Required rights are assigned on a "class" basis and not on an "instance" basis. Hence, no change to existing object interfaces is necessary to assign required rights. But also all instances of a class, therefore, will always have the same required rights.

The AccessDecision object determines the validity of invocation requests relying on the privilege and control attributes provided by AccessPolicy and RequiredRights objects. There are no explicit rules on how to calculate the access decision. Derived from the examples given in document [10], the following (partial) rule can be defined: A requestor will be granted authority to execute method m of object o if the credential associated with the request contains a privilege attribute that grants a right into a domain containing object o and if the same right is also included in the RequiredRights (RR) table of method m. In other words, method m will be executed if the requestors's rights match *at least* one of the rights specified in the method's entry of the RequiredRights table. For example, a user who is granted any right in a method's RR table entry is authorized to invoke the method.

User	**Privilege Attribute**
u_1	a_1, g_1
u_2	a_2, a_3
u_3	g_2

Object	**Domain**
$o_1, o_2, o_5, o_8, o_{12}$	d_1
o_8, o_9	d_2

Objects	**Class**
o_1, o_8	c_1
o_2, o_5	c_2
o_{12}	c_3

Table 1: Groupings

Let Fig. 2 define the access policy of domain d_1 and let Fig. 3 prescribe the required rights for classes c_1, c_2, and c_3. In the context of the grouping provided in Tbl. 1, CS authorization gives access granted by the AccessPolicy of domain d_1 in at least the following cases:

- u_1 can execute methods m_1 and m_2 of objects o_1, o_8 as an initiator but may execute only method m_2 as a delegate.

- u_1 can execute methods m_1 and m_2 of objects o_2, o_5 as an initiator but may execute no methods of o_2 and o_5 as a delegate.
- members of group g_2 can execute method m_2 of objects o_2, o_5 as an initiator but may execute no methods of o_2 and o_5 as a delegate.

In CS [10, p. 143], it is left open how an AccessPolicy object combines rights granted by different Privilege Attribute entries, in the case that a subject has more than one privilege attribute to which the AccessPolicy grants rights. As mentioned above, there are no composition rules for domain hierarchies either. This undefinedness leads to different interpretations and, therefore, to different access control decision functions (see Section 4.2).

With respect to the above scenario, for example, it is not clear whether u_1 as a delegate may execute method m_1 of object o_{12} if the request carries privilege attributes a_1 and g_1. As privilege attribute a_1 does not grant access, a POSIX-based access decision function would deny access. Thus, to conform to POSIX, the access decision function has to implement an evaluation order on privilege attribute types or on overlapping domains. This would also enable one to realize some kind of exception handling policy, e.g. "all members of group X except user u".

CS authorization enables the formulation of access control policies where a role is combined with a group membership, e.g. "Only somebody who is a manager and a member of project-X is allowed to change project X's time schedule".

3 A Formal Model of Authorization

We use Lampson's access matrix model, as defined by Harrison, Ruzzo, and Ullman [4] but in a less restricted form. The model makes use of two concepts, the *protection system* and its *state*, the latter represented by an access matrix, in which a set of *subjects* (e.g. users) is granted *rights* to perform operations on a set of *objects*.

A protection system consists of a finite collection of commands to change the protection state. The execution of a command is controlled by a boolean condition on the current contents of the access matrix. For each command, the authority to execute as well as its effect on the protection state have to be specified. The scheme is defined by the security administrator when the system is first set up and thereafter remains fixed.

Definition 1. A protection system consists of a finite set $\mathcal{R}$ of generic rights and a finite set of commands of the form:

$$
\begin{array}{l}
\textbf{command } \alpha(X_1, X_2, \ldots, X_k) \\
\quad \textbf{if } r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots \\
\quad \textbf{then } op_1,\ op_2,\ \ldots \\
\textbf{end}
\end{array}
$$

Here, α is a name and $X_1, \ldots, X_k$ are formal parameters, each X_i being either a subject or an object. Subscripts s, s_i, o, and o_i are integers in the range $1 \ldots k$ indicating whether a parameter is a subject or an object. Each r_i is a right and each op_i is one of the primitive operations

> **enter** r **into** $[X_s, X_o]$
> **delete** r **from** $[X_s, X_o]$
> **create subject** X_s
> **create object** X_o
> **destroy subject** X_s
> **destroy object** X_o

whereby each s and o is an integer in the range $1 \ldots k$, k being the number of parameters in the command, X_s denoting a subject, and X_o denoting an object.

Definition 2. A state of a protection system is a triple $\langle S, O, M \rangle$ where S is the set of subjects, O is the set of objects and M is an access matrix with a row for every subject in S and a column for every object in O. $M[s, o]$ is a subset of rights to object o possessed by subject s.

The set of commands lets the state evolve by the autonomous activity of the subjects. Each command may specify a test for the presence of certain rights in certain positions of the matrix verifying that the action to be performed by the command is authorized. Each operation defines some modification to be made to the access matrix.

4 A Model of CS Authorization

In this section, we develop a specification that defines the semantics of authorization in the CS model as described in Section 2.

4.1 Structure

Let $\mathcal{R}$ be the set of rights; elements of $\mathcal{R}$ are denoted by r. RightsFamilies subdivide $\mathcal{R}$ into sets of rights types. Not all subsets of $\mathcal{R}$ are legal rights types. We require that $\mathcal{R}$ contains at least the rights type $corba = \{\texttt{g}, \texttt{s}, \texttt{m}\}$. The distinguished symbol $\surd$ shall be member of $\mathcal{R}$.

Let $\mathcal{U}$ denote the set of principals or users, the active entities in the system. Authorization units, however, are privilege attributes. Let $PA(u)$ denote the set of privilege attributes assigned to user u. The set $\mathcal{P}$ of privilege attributes may be subdivided, for example, into the set of access identifiers, user groups, and roles. The relationship between users and privilege attribute types is defined outside of the CS specification. For example, if there is a unique access identifier for each user, the set of users and the set of access identifiers may coincide. A privilege attribute type like group or role may also possess an internal structure, e.g. a hierarchical order that could be employed in the access decision function.

As an example of the use of privilege attributes, we consider two kinds of privilege attribute types, the set of access identifiers $\mathcal{A}$ and the set of user groups $\mathcal{G}$. Let $\mathcal{O}$ denote the set of objects and let $\mathcal{D}$ denote the set of domains. We assume a relation *domain* (3) that assigns objects to domains. We denote the set of object classes by $\mathcal{C}$ and require that an object has at most one class. The relation *class* (4) gives the class to which an object belongs.

$$\begin{aligned} \mathit{accId}: &\quad \mathcal{A} \times \mathcal{U} && (1)\\ \mathit{usrGrp}: &\quad \mathcal{G} \times \mathcal{U} && (2)\\ \mathit{domain}: &\quad \mathcal{O} \times \mathcal{D} && (3)\\ \mathit{class}: &\quad \mathcal{O} \times \mathcal{C} && (4) \end{aligned}$$

We require that relation *accId* uniquely identifies the user associated with each access identifier, i.e. there is no access identifier shared by two users:

$$\forall a \in \mathcal{A},\ \forall u_1, u_2 \in \mathcal{U}:\ (\langle a, u_1 \rangle \in \mathit{accId} \wedge \langle a, u_2 \rangle \in \mathit{accId}) \Rightarrow u_1 = u_2 \qquad (5)$$

As the CS specification imposes no structure on a *usrGrp* relation, e.g. groups may be members of other groups, we model privilege attribute types as flat structures.

To reduce the cardinality of the set of method names, we number the methods of each class: $m_1, m_2, \ldots$. Let $\mathcal{M}$ denote this set whose cardinality is determined by the class with the largest number of methods. If $\mathcal{N}$ is the set of all methods, function $\mathit{name} : \mathcal{N} \times \mathcal{C} \to \mathcal{M}$ returns the index of a method with respect to a given class. By means of $\langle m, c \rangle$, where m is the index of the method and c is the name of the class to which the method belongs, we uniquely identify a method.

With the above description of the elements of the CS model we now can give a formal definition of its structure.

Definition 3. A CS state parameterized by a set $\mathcal{R}$ of rights is a triple $\langle S, O, M \rangle$ where $S = \mathcal{P} \cup \mathcal{O} \cup \mathcal{M}$ is the set of subjects, $O = \mathcal{U} \cup \mathcal{D} \cup \mathcal{C}$ is the set of objects and M is an access matrix.

It is interesting to note that in the CS access matrix target objects occur as subject entries and users occur as object entries. This shows clearly that authorization in CORBA Security refers to privilege attributes (security subjects) and domains (security objects). Combining all authorization information into a single matrix makes consistency checks easier.

Fig. 4 illustrates the structure of a CS state. It defines an access policy for domain d_2 and also uses a second RightsFamily, $\mathit{other} = \{\texttt{g},\texttt{u},\texttt{o},\texttt{m},\texttt{t},\texttt{s}\}$. For space limitations, symbol '&' shall denote the 'All' rights combinator and symbol '|' shall denote the 'Any' rights combinator. Moreover, rights are not prefixed by the name of their RightsFamily. In the

Objects Subjects	u_1	u_2	u_3	d_1	d_2	c_1	c_2	c_3
a_1	$\surd$			`gs-` `-u-m-s`	`---` `------`			
a_2		$\surd$		`---` `------`	`g--` `----t-`			
g_1	$\surd$			`g--` `-u---s`	`gs-` `-u----`			
g_2			$\surd$	`gs-` `------`	`---` `------`			
o_1				$\surd$		$\surd$		
o_2				$\surd$			$\surd$	
o_5				$\surd$			$\surd$	
o_8				$\surd$	$\surd$	$\surd$		
o_9					$\surd$		$\surd$	
o_{12}				$\surd$				$\surd$
m_1						`&:-s-`	`&:gs-`	`|:g-om-s`
m_2						`|:gs-`	`&:---m-s`	`|:---`

Figure 4: The access matrix of CS Authorization

matrix, partition *Access Identifiers* $= M[\mathcal{A},\mathcal{U}]$ defines relation (1), partition *User Groups* $= M[\mathcal{G},\mathcal{U}]$ defines relation (2), and partition *Domains* $= M[\mathcal{O},\mathcal{D}]$ defines relation (3). Partition $RR = M[\mathcal{N},\mathcal{C}]$ defines the rights required to invoke a method, and partition $GR = M[\mathcal{P},\mathcal{D}]$ defines the set of rights granted to privilege attributes. Let function $RR(m)$ give the set of required rights to invoke method $m = \langle n, c\rangle$.

Using this matrix, we can also give a definition of the *accId* relation (1), the user group relation (2), the domain relation (3), and of the class relation (4):

$$\begin{aligned} \mathit{accId}(u) &= \{a \mid \surd \in [a,u]\} \\ \mathit{usrGrp}(u) &= \{g \mid \surd \in [g,u]\} \\ \mathit{domain}(o) &= \{a \mid \surd \in [o,d]\} \\ \mathit{class}(o) &= \{c \mid \surd \in [o,c]\} \end{aligned}$$

Let $PA(u) = \mathit{accId}(u) \circ \mathit{usrGrp}(u) \circ \ldots$ be the set of privilege attributes assigned to user u. To ensure that an access decision function (see below) always returns a valid result, a CS state has to satisfy certain constraints.

Definition 4. A well-formed CS state is a CS state $\langle S, O, M\rangle$ in which the access matrix satisfies the following condition:

$$\forall o \in \mathcal{O}\ \exists d \in \mathcal{D}\ :\ d \in \mathit{domain}(o)$$

The above condition expresses that each access-controlled object in a CORBA system must be a member of at least one domain [10, p. 143].

4.2 Access Control Decision

In this subsection, we define how access decisions are taken. In Fig. 5, links relate the elements of the access control policy as expressed in Figs. 2 and 3 (CS tables) and in Fig. 4 (access matrix). In Fig. 5, the structure of privilege attributes and domains is flattened, e.g. a domain refers to all objects that transitively belong to it.

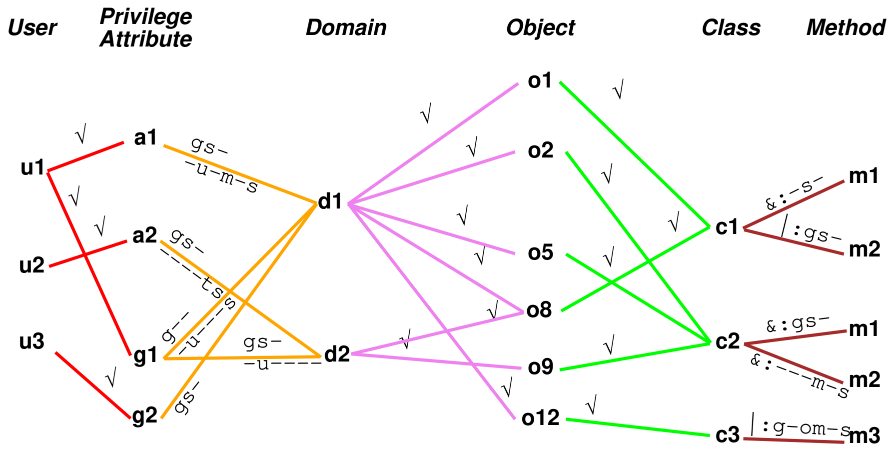

Figure 5: Associations between the CS authorization elements

If we assume that an object invocation request carries all privilege attributes of the initiating user, the task of an Access Decision Function could then be described as follows: check whether the requesting user is connected with the demanded method via the object on which the method should be executed. In the case of an OR semantics, the existence of one path labeled twice with the same right (neglecting $\surd$) would be sufficient to grant the request.

Disjunctive Semantics (AnyRight)

CS grants access to user u for an object o by method m if there exists a right r which occurs in a cell belonging to a privilege attribute p and a domain d and the same right occurs in a cell belonging to method m and the class of object o. The privilege attribute p must be included in the user's request and can be an access identifier or a group membership.

Object o must be a member of domain d.

Definition 5. An OR-based access decision function permits a request of user u to execute method m on object o if it holds that:

$$\begin{aligned}&\exists p, c, d, r : p \in \mathcal{P}, c \in \mathcal{C}, d \in \mathcal{D}, r \in \mathcal{R} :\\ &\quad p \in \mathit{PA}(u) \wedge c \in \mathit{class}(o) \wedge d \in \mathit{domain}(o) \wedge r \in [p, d] \wedge r \in [m, c]\end{aligned} \tag{6}$$

Note that the above formula requires the existence of at least one right in both the GR table entry and the RR table entry in order to allow access, i.e. if the RR table entry is empty (`|:---`), access is denied to everybody.

As all sets in Formula 6 are finite, we can completely express the decision function in propositional logic:

$$\bigvee_{p\in\mathcal{P}} \surd \in [p, u] \wedge \ (\bigvee_{c\in\mathcal{C}} \surd \in [o, c] \wedge \ (\bigvee_{d\in\mathcal{D}} \surd \in [o, d] \wedge \ (\bigvee_{r\in\mathcal{R}} r \in [p, d] \wedge r \in [m, c]))) \tag{7}$$

In Formula 7, disjunction is over privilege attributes, classes, domains, and rights, making the underlying "or" semantics explicit. A user receives the union of the rights of the privilege attributes she owns, a method receives the union of the rights of the domains to which it belongs, and an RR table entry ORs all its rights.

Conjunctive Semantics (AllRights)

If the access permission depends on all rights set in the RR table entry, the access decision function must compute the AND semantics as follows:

$$\bigvee_{p\in\mathcal{P}} \surd \in [p, u] \wedge \ (\bigvee_{c\in\mathcal{C}} \surd \in [o, c] \wedge \ (\bigvee_{d\in\mathcal{D}} \surd \in [o, d] \wedge \ (\bigwedge_{r\in\mathcal{R}} r \in [m, c] \Rightarrow r \in [p, d]))) \tag{8}$$

Note that the above formula permits any user to execute a method if the method's RR table entry is empty (`&:---`). However, it requires further that there be one domain d that grants all the necessary rights to requester u.

To allow for the most liberal case where (granted) rights can be cumulated through different privilege attributes (access identifier/group) and through different domains, the access control function has to be adapted as follows:

$$(\bigvee_{c\in\mathcal{C}} \surd \in [o, c] \wedge \bigwedge_{r\in\mathcal{R}} r \in [m, c]) \Rightarrow (\bigvee_{p\in\mathcal{P}} \surd \in [p, u] \wedge \ (\bigvee_{d\in\mathcal{D}} \surd \in [o, d] \wedge r \in [p, d]))$$

Although it might be debatable whether it is of practical use to combine access rights gained through different means, e.g. user group membership and role association, we employ this particular access policy in order to be able to implement the access policy expressed at the end of Section 2.

Definition 6. An AND-based access decision function permits a request of user u to execute method m on object o if it holds that:

$$\exists c : c \in \mathcal{C} : \ \forall r \in \mathcal{R} : \ c \in \mathit{class}(o) \wedge r \in [m, c] \Rightarrow$$
$$\exists p, d : p \in \mathcal{P}, d \in \mathcal{D} : \ p \in \mathit{PA}(u) \wedge d \in \mathit{domain}(o) \wedge r \in [p, d]$$

Compared with OR semantics, computation complexity is greater as one has to check for each right in the RR table whether the same right occurred in the GR table. However, if CS authorization should be implemented using DCE, an AND semantics fits well. Otherwise, the CS authorization oracle would have to call the DCE access control function, e.g. `dce_acl_is_client_authorized`, for each right in the GR table entry until a match is found.

4.3 Administration Interface

For security administration, CORBA Security defines a basic interface for the maintenance of Domain Access Policies and Required Rights. However, there are no interfaces to manage domain membership and the structure of domains. The management of privilege attributes is left open, too. To complete the definition of CS authorization, we briefly discuss how administration operations could be given a semantics as well.

The execution of any CS administration operation modifies the state of the access matrix. Thus, they have to be modelled as commands of the protection system (see Def. 1). On the other hand, these operations are subject to access control as well and should occur as methods listed in the subject dimension of the access matrix. However, in their most abstract form, not prescribing any particular implementation, the administration operations would merely occur as commands.

In CS authorization, ownership is separated from administration, i.e. authorization data itself is not protected by special rights, e.g. the control right in DCE. To show how administration operations can be managed under this paradigm, we assume that there are special protection objects, one for each cell of the access matrix. However, we leave it open how these protection objects are grouped into domains, one for each set of 'cell objects' controlled by the same access policy. For example, the domain PGR could define the access policy for the administration of partition GR if the power of granting rights is controlled by only one access policy.

As an example of methods that change the contents of the matrix, the command below grants right g (to perform get methods on objects that are members of domain d) to the privilege attribute named p:

command $\texttt{grantRight}_{\texttt{g}}(s_A, p, d)$

if m$\in[s_A$,PGR]
then enter g **into** $[p, d]$
end

In the above command, the security administrator s_A is authorized to perform this command if she, for example, possesses right m on domain *PGR*. However, for this to work, we have to make the implicit assumption that the first parameter of a command always names the requestor of the method.

5 Mandatory Access Control Policies in CS

In this section we implement a label-based access control policy to demonstrate the expressivity of CS authorization. First, we introduce another rights type called *labelRights*. Assuming three security levels, this rights type consists of the individual rights r_1, r_2, r_3, w_1, w_2, and w_3. For example, right r_3 expresses a higher security level than r_2.

Security labels are attached to subjects and objects. A label on subjects is called a security clearance. A label on objects is called a security classification. We model security clearances by creating a domain called *LABEL* and grant each subject label rights according to her clearance. For example, if subject s has clearance 2 she gets rights r_1, r_2, w_2, and w_3, denoting that she can read all objects of classification 2 or lower, and that she can write on all objects with the same classification or higher.

As access control granularity in CS is on the method level, we have to determine the classification level of each method. For all MAC schemes it is assumed that each method has either read or write characteristics. These levels are written into the RR table. All objects are members of domain *LABEL*.

Objects **Subjects**	u_1	u_2	u_3	*LABEL*	c_1	c_2	c_3	c_4
a_1	$\surd$			r1,w1,w2,w3				
a_2		$\surd$		r1,r2,w2,w3				
a_3			$\surd$	r1,r2,r3,w3				
o_1				$\surd$		$\surd$		
o_2				$\surd$			$\surd$	
o_5				$\surd$	$\surd$			
o_8				$\surd$				$\surd$
m_1					w2	r2	w3	w1
m_2					r3	w1	r1	w2

Figure 6: Mandatory Access Control in CS

Assume the above subject s wants to invoke method m_1 where $RR(m_1) = \{r_1\}$. Access is allowed as the right r_1 is granted to subject s as well as being included in the RR table of method m_1. Assume further that we have $RR(m_2) = \{w_1\}$ and $RR(m_3) = \{w_3\}$. Then the same subject s is allowed to invoke method m_3 but is not allowed to invoke method m_2 (no write-down, *-property). Note that this scheme works with both OR and AND semantics.

Fig. 6 gives an example of a CS matrix expressing a mandatory access control policy. Note that there is only one domain named $LABEL$. For example, method m_1 of class c_2 can be executed by subject a_2 as this subject has clearance level 2 and therefore possesses label right r_2. Subject a_1 with clearance level 1 is not allowed to execute this method but may execute method m_2 of class c_4 (write-up).

A DAC policy augmented with mandatory access control to enforce information flow policies could be implemented as well. There is a (simple) solution for an AND-based access decision function and a more complex solution for an OR-based access decision function using the technique illustrated in [12].

6 Conclusion

We presented a formalization of the authorization scheme in CORBA Security. By providing an access matrix, we defined the structure and the consistency requirements of an CS authorization state. The definition of associated access decision functions showed clearly the underlying union semantics of CS authorization. A set of commands defined the semantics of the CS interface. CS access policy enforcement is independent of a particular policy. We showed that multilevel operations can be implemented as requested in [3]. The implementation of compartmented-mode operations, however, would require extensions.

The current definition of CS authorization does not address the specification of default access rights for newly created objects. Although CS authorization has been specified in an object-oriented paradigm, the current document [10] gives no advice on how to beneficially administer object-oriented data systems [13].

Compared with other authorization models [1], in particular those designed for object-oriented database systems, CS authorization lacks the notions of composite and compound subjects, hierarchies (of groups, roles, domains, rights), and negative rights. These models allow one to derive (implicit) authorizations from the relationships among subjects, objects (and in particular along the class inheritance hierarchy), and rights, mostly by providing explicit derivation rules. A rich structure is therefore beneficial. However, a tradeoff always exists between the complexity of the derivation system and the understandability for a human being.

To foster analysis of secure CORBA systems it would be interesting to see how much of our access matrix model could be mapped on Sandhu's Typed Access Matrix model [11].

Acknowledgments

The author wishes to thank M. Benantar, G. Blakley, and A. Nadalin for sharing their insights into CORBA Security. Special thanks go also to M. Dacier, M. Steiner, and M. Waidner for their helpful comments on drafts of this paper.

References

[1] E. Bertino, S. Jajodia, and P. Samarati. Access controls in object-oriented database systems: Some approaches and issues. In N. Adam and B. Bhargava, editors, *Advanced Database Concepts and Research Issues*, volume 759 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[2] M. Benantar, G. Blakley, and A. Nadalin. Approach to object security in Distributed SOM. *IBM Systems Journal*, 35(2):192–203, 1996.

[3] S.L. Chapin, W.R. Herndon, L. Notargiacomo, M.L. Katz, and T.J. Mowbray. Security for the common object request broker architecture (CORBA). In *10th Annual Computer Security Applications Conference*, pages 21–30, 1994. Computer Society Press.

[4] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[5] B. Hartman, D. Nessett, and N. Yialelis. Scalability of Security in Distributed Object Systems: Panel Session. In *The Internet Society 1996 Symposium On Network And Distributed System Security (NDSS '96)*, pages 40–41, 1996. Computer Society Press.

[6] Object Mangagement Group.OMG White Paper on Security. Document 94.4.16, OMG Security Working Group, April 1994.

[7] Object Mangagement Group. Object Services RFP 3. Document 94-7-1, 1994.

[8] Object Mangagement Group. Object Management Architecture Guide. Revision 3.0, June 1995. John Wiley & Sons.

[9] Object Mangagement Group. The Common Object Request Broker: Architecture and Specification. Revision 2.0, July 1995.

[10] Object Mangagement Group. CORBA Security. Document 95-12-1, December 1995.

[11] R.S. Sandhu. The typed access matrix model. In *IEEE Symposium on Research in Security and Privacy*, pages 122–136, 1992. Computer Society Press.

[12] R.S. Sandhu and S. Ganta. On the minimality of testing for rights in transformation models. In *IEEE Symposium on Research in Security and Privacy*, pages 230–241, 1994. Computer Society Press.

[13] D.L. Spooner. The impact of inheritance on security in object-oriented database systems. In C.E. Landwehr, editor, *DATABASE SECURITY II: Status and Prospects*, pages 141–150. Elsevier Science Publishers B.V. (North-Holland), 1989.