Implementation of Multiple Access Control Policies within a CORBASEC framework

Ramaswamy Chandramouli
Computer Security Division, ITL
NIST, Gaithersburg, MD 20899
(Chandramouli@nist.gov)

Abstract

This paper presents an architecture for implementation of multiple access control policies within a CORBASEC framework. CORBASEC is a security service specification provided by OMG for implementation of security in CORBA compliant systems. Because of the differences in ORB implementations among vendors and divergent security needs of the CORBA based applications in different commercial markets, CORBASEC provides a general *security reference model.* The reference model can then be used by implementers to customize a security model depending upon their system and application needs. We have extracted one such model from the CORBASEC security reference model (or framework) to be used in an architecture whose objective is to support multiple access control policies within a single unified system. Throughout this paper we refer to this architecture using the acronym MACP. The central component of MACP is an Access Computation Server (ACS) which hosts multiple modules representing different security policies. These modules are implemented using an expressive logic based “Authorization, Policy Rules Specification & Decision Language” (APRS-DL) which provides constructs for capturing the access attributes and policy rules associated with several well known access control policies. We illustrate the use of APRS-DL for generation of access control decisions for a Discretionary Access Control (DAC) policy using the Role Based Access Control (RBAC) authorization management concept.

1. Introduction

Providing security in distributed systems is complicated by the fact that each of the subsystems supports different security technologies. Specifically, in the area of access control, each of the subsystems may support different access control enforcement mechanisms. The access control attributes that these mechanisms use, in turn, dictate the set of access control policies that can be implemented using these mechanisms. In many instances, the access control policy is built into the access enforcement mechanism. Consequently, the set of access control policies that can be defined for a subsystem within a large distributed system, is constrained by these mechanisms. However, we know from our practical needs that any one policy cannot meet the protection needs of various types of data in an organization. Hence we are faced with a situation where an organization has to manage an Enterprise Access Control Policy that stipulates different set of access control policies for different application domains and each one of them are implemented using a different access control technology. The above scenario underscores the need to decouple the *policy definition and access computation process* from the policy enforcement mechanism. One of the early architectures to attempt this decoupling is given in [ncsc93]. The basic requirement to enable this decoupling is that there is a good policy translator (or an Access Computation Engine) which allows the definition of rules and access attributes pertaining to any policy and has the capability to process those rules and generate access authorization/denial messages. The access message generation logic is thus made independent of the enforcement mechanism and the same mechanism can be used whatever the access control policy that needs to be supported. Hence the goal is to design an Access Computation with multiple policy support. Towards this objective Jonscher and Dittrich [jd96] proposed an access control system for protection of information in distributed federated database systems which allows the enforcement of different policies. However, their system requires the definition of multiple reference monitors, each enforcing a particular access policy. To alleviate this problem, Jajodia et al [oak97, sigmod97] have proposed a flexible authorization manager (FAM) that will enforce multiple access policies within a single unified system.

In this paper we have developed an architecture (with acronym MACP) for definition and enforcement of multiple access control policies in the context of a CORBASEC security model. The central component of MACP is an Access Computation Server (ACS) which hosts multiple modules representing different security policies. These modules are implemented using a expressive logic based "Authorization, Policy Rules Specification & Decision Language" (APRS-DL). The organization of this paper is as follows: In section 2 , we give a brief overview of the CORBASEC specification and its underlying philosophy. The CORBASEC specification (referred to as CORBASEC framework in the rest of the paper) provides a security reference model so that implementers can choose a particular security model (from the various components presented in the reference model) depending upon their system and application needs. In section 3, we describe the particular security model that we have chosen from the CORBASEC framework in terms of parameters that are relevant from an access control viewpoint. We also describe in this section, the various building blocks of the MACP architecture that will implement multiple security policies within the chosen CORBASEC security model .In section 4, we outline the implementation aspects of MACP. In section 5, we present the conclusions.

2. **CORBASEC – a security reference model for CORBA Security**

CORBA presents a set of specifications for developing distributed object-based systems on heterogeneous platforms (different language and operating systems). The transparency in distribution is achieved through the use of software entities called ORBs (Object Request Brokers) which forward and mediate all object requests (invocation of methods on objects).

The OMG which has published the CORBA specifications has also published a set of specifications called CORBASEC [CSEC97] which provides a *security reference model* for developers of CORBA compliant systems.

The CORBASEC security reference model is technology neutral since it has to provide the necessary freedom of implementation for ORB manufacturers as well as the application developers in a CORBA environment. However the model specification covers various security functionality. These are: (a) Identification & Authentication (b) Authorization & Access Control (c) Security Auditing (d) Security of Communication (e) Non-Repudiation and (f) Security Management & Administration. Since our objective in this paper is to outline our ideas regarding the implementation of multiple access control policies within a CORABSEC framework, we are only interested in those features & facilities within the CORBASEC security reference model that provide for Authorization & Access Control functionality.

Even within the realm of access control functionality, different combinations of features & facilities results in different security models. We have chosen one such model based on the following parameters. They are: (a) the layer at which access control policy is enforced (System or Application layer) (b) the access authorization model used for access computation (deciding whether an operation is to allowed or denied) and (c) choice of a package for providing access control services.

2.1 Choice of Access Control Policy Enforcement Layer

CORBASEC provides for enforcement of access control policy at two layers. At the *System layer* it is enforced by the distributed ORB and the security services it uses, for all applications, whether they are aware of security or not. This aspect of the policy is termed the Object Invocation Access Policy and governs whether this client, acting on behalf of the current principal, can invoke the requested operation on this target object.

The access policy enforced at the *Application layer* is termed the Application Access Policy and is enforced within the client and/or the target object. This policy can be concerned with controlling access to object's internal functions and data, or applying further controls on object invocation.

2.2 Access Authorization Model

The CORBASEC framework outlines an authorization model based on the use of access decision functions for generation of ALLOWED/DENIED messages at any layer (System or Application). The general functionality of these Access Decision Functions is to decide whether a particular user invoked action can be allowed or not, by applying access control rules to one or more of the following categories of information:
(a) Action (Operation) and Context Information
(b) Initiator Privilege Attributes
(c) Control Attributes of the Target Object.

The access policy is thus built into these functions. The role of Access Decision Function in the CORBASEC framework is illustrated through Figure 2.1 below:

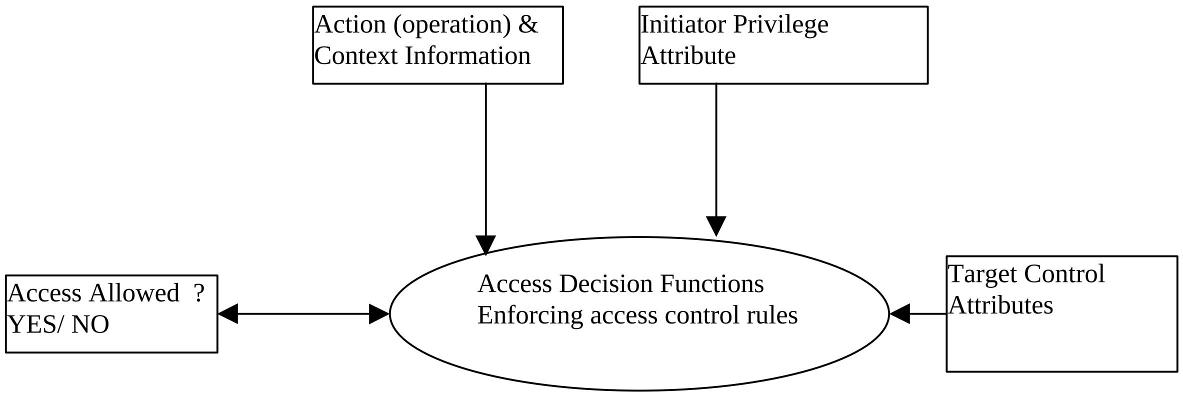

Action (operation) & Context Information
Initiator Privilege Attribute
Access Allowed ? YES/ NO
Access Decision Functions Enforcing access control rules
Target Control Attributes

FIG 2.1 Role of Access Decision Functions in the CORBASEC framework

2.3 Choice of Packages for access control services

The CORBASEC framework enumerates a list of " feature packages" that can be used with an ORB to provide various security services. There are two categories of "feature packages". The first category of packages provide the basic security functionality whereas the second category of packages provide different levels of secure interoperability. The packages in the first category in turn can be divided into (a) Security Feature Packages and (b) Security Replaceability packages. CORBASEC specification v1.2 (Page 15-12) states that an ORB can provide security by implementing a Security Feature Package into it or it can make use of any of the facilities provided by Security Replaceability Packages. It also clearly states that by implementing the Security Feature Package into an ORB, we are limited to the standard access policy specified in the CORBASEC and that we do not have the flexibility to choose an access control policy. The CORBASEC specification also categorically states that such a flexibility (in the choice of access control policies) is only possible by implementing at least one of the Security Replaceability Packages.

3. Architecture for Multiple Access Control Policies for CORBASEC (MACP)

As already stated, the architecture for supporting multiple access control policies within a CORABSEC framework (acronym MACP) developed in this paper, is based a specific security model (based on access control view) extracted from CORBASEC framework. The features of this security model are described in Section 3.1. The building blocks of the MACP architecture that uses this security model are given in Section 3.2. Sections 3.3 through 3.5 describe the functionality and role of each of these building blocks of MACP.

3.1 Underlying CORBASEC Security Model for MACP

We now describe the underlying CORBASEC security model upon which MACP has been built in terms of the features chosen from the three CORBASEC framework parameters discussed through sections 2.1 through 2.3.

3.11 Choice of Policy Enforcement Layer: In MACP architecture, we have chosen to provide access control at *System* layer. The following are the advantages of System layer access control as opposed to Application layer access control:

(a) In a distributed object environment, there may be many different types of objects and each object in turn may be used by multiple applications (utilizing the object re-use concept). In such an environment, a better security assurance can be obtained by implementing access control on these objects at the system level.

(b) Access to different objects within a single application system may be governed by different access control policies. For this scenario, providing access control at the system level will enable generation of different types of client credentials for an application client, based on the identity of the client as well as the object the client is invoking.

(c) A third advantage of providing access control at the system level is that, the application objects need not be aware of security and hence can be ported to environments that enforce different access control policies and use different enforcement mechanisms.

In spite of the above advantages, it must be mentioned that a finer level of access control can be provided at the application layer as each application object can control access to its own data & functions based upon the value of parameters in the user object invocation request.

3.12 Choice of Access Authorization Model: In MACP architecture , we have adopted the generic access authorization model based on access decision functions. However in order that these functions provide authorizations based on a wide range of access control policies, it is necessary that these are implemented in a flexible logic based language which can handle different kinds of access attributes. The APRS-DL (Authorizations Policy Rules Specification & Decision Language) described in this paper can meet this requirement by being expressive enough to incorporate client security attributes (also called privilege attributes) as well as target object control attributes in addition to contextual information.

3.13 Packages for providing access control services: Our goal behind the development of an architecture in this paper is not only to have flexibility in the choice of access control policies but also to support multiple policies as well. Hence in tune with our objectives, the CORBASEC security model used for our architecture should contain one of the Security Replaceability packages rather than any Security Feature Packages. The CORBASEC specification provides two replaceability packages. They are: (a) ORB Services Replaceability Package and (b) Security Service Replaceability Package. The

Security Service Replaceability Package is the one that is included in our architecture since this is the one that provides a set of interfaces for invoking the needed security services without the services needing to know how the ORB works (for example , how the required policy objects are located), so that they can be replaced independently of that knowledge. In addition the package is used by the ORB only for access decision messages and hence it is free to use any access enforcement mechanism to carry the out the logic behind the message. In other words this particular package implementation enables the decoupling of access control enforcement mechanism from access computation process. This package is implemented in the MACP architecture in an entity called the "**Access Computation Server**". The Access Computation Server (ACS) provides a set of interfaces that provide the necessary access control services needed by client side and target side ORBs.

In summary, the CORBASEC security model that MACP has adopted is meant to facilitate support for multiple access control policies at the system layer with all the requisite access control services for ORBs being provided by the Access Computation Server which contains multiple access decision modules implemented using APRS-DL.

3.2 Building Blocks of MACP Architecture

An underlying philosophy behind the development of MACP architecture is the elimination of tight coupling between access enforcement mechanism and access computation process. The software entities involved in MACP are shown in Fig 3.1. The functionality of these entities and the conceptual framework on which they are based are described below:

(a) *Access Authorization Model (AAM)*: To implement any policy a logical framework called Access Authorization Model (AAM) is required. As already stated in section 3.12, the AAM in MACP is based on access decision functions implemented in APRS-DL. The required characteristics for such a AAM for inclusion in MACP are given in section 3.3

(b) *Access Computation Server (ACS)*: The MACP requires a server entity with interfaces that support the following: (a) create & define modules for supporting different access control policies (b) Process client requests under various policy frameworks and generate access control decisions. This entity in MACP is called the Access Computation Server and is built in two components (one for the use of client side ORB and the other to be used by the target side ORB- we call the ORB on the client side as **client side ORB** and the ORB at the site where target object is located as **target side ORB**). The role of ACS at the client side ORB (called ACS/Cl_ORB) is to generate client credentials associated with a client request and the role of ACS at the target side ORB (called ACS/Tr_ORB) is to generate access decision based on client credentials and target object control attributes. The detailed functionality of these ACS components are described in section 3.4.

(c) *Authorization Policy Rules Specification & Decision Language (APRS-DL)*: The generation of client credentials at the client side ORB (through ACS/Cl_ORB) and generation of access decision messages at the target side ORB (through ACS/Tr_ORB) are the output of functions are called Access Decision Functions implemented in these respective ACS components. These Access Decision Functions are implemented using a Authorization Policy Rules Specification & Decision Language (APRS-DL) . The APRS-DL constructs and capabilities are described in detail in section 3.5.

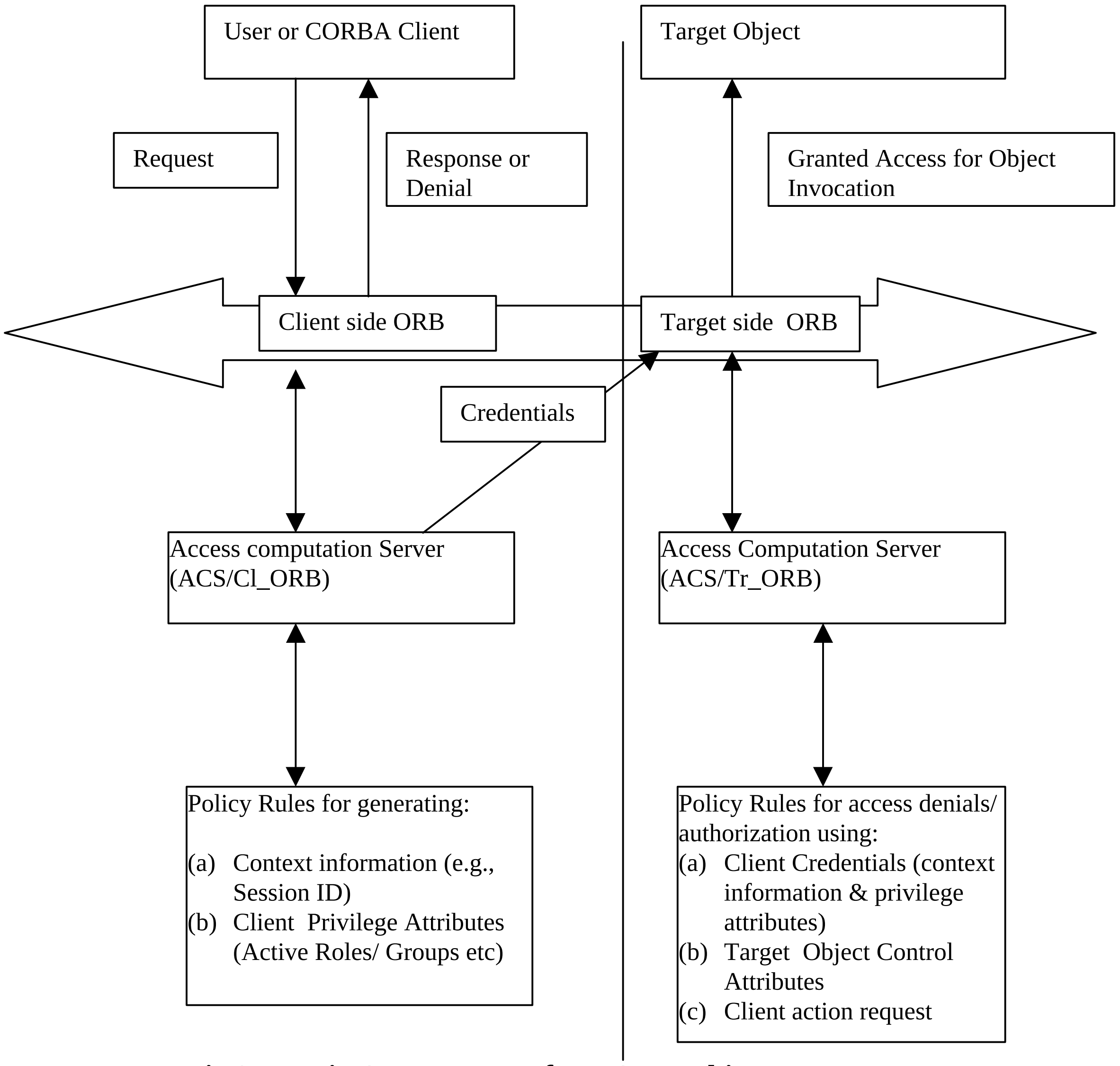

User or CORBA Client
Target Object
Request
Response or Denial
Granted Access for Object Invocation
Client side ORB
Target side ORB
Credentials
Access computation Server (ACS/Cl_ORB)
Access Computation Server (ACS/Tr_ORB)
Policy Rules for generating:
(a) Context information (e.g., Session ID)
(b) Client Privilege Attributes (Active Roles/ Groups etc)
Policy Rules for access denials/ authorization using:
(a) Client Credentials (context information & privilege attributes)
(b) Target Object Control Attributes
(c) Client action request

Fig 3.1 Basic Components of MACP Architecture

3.3 Access Authorization Model (AAM)

In general an access authorization model is a framework used for processing access control rules pertaining to a policy to arrive at a decision to allow or deny an operation request. The AAM in MACP is based on the use of access decision functions implemented in APRS-DL whose constructs can capture policy attributes pertaining to mutiple policies. Further the first order predicate logic based APRS-DL also guarantees the following properties for AAM. They are:

(a) Completeness – For every access request, a grant or deny decision should be available
(b) Consistency – Taking into account contextual information, the grant or denial decision should the same across multiple access requests, unless the access policy itself is redefined.

3.4 Access Computation Server (ACS) architecture

The ACS is made up of two components: (a) ACS/Cl_ORB component and (b) ACS/Tr_ORB component. Both of these components have associated with them a set of Access Decision Functions.

The interfaces to those access decision functions in ACS/Cl_ORB component are invoked by the client side ORB to generate the requisite client credentials associated with a client request. These credentials are then passed on to the target side ORB. The client credentials generated by access decision functions in ACS/Cl_ORB consists of the following information: (a) Context information (e.g., time of access, Session ID etc.,) and (b) Client privilege attributes (e.g., the roles/groups to which the client belongs).

The target side ORB then invokes the set of interfaces associated with Access Decision Functions implemented in ACS/Tr_ORB in order to grant/ or deny the client request on the target object. The access decision functions that form part of ACS/Tr_ORB processes the following types of information:

(a) Client or Principal privilege attributes
(b) Context Information
(a & b are components of client credentials received from client side ACS)
(c) The control attributes of the target object
(d) Object invocation request consisting of the object, method invoked and parameters.

3.4.1 Supporting Multiple Access Control Policies within ACS

As already stated the Access Computation Server is the entity that houses the access decision functions. Since these functions are implemented in first order predicate logic based APRS-DL language, they are capable of capturing policy attributes pertaining to different access control policies. Hence ACS can house multiple access decision modules each one providing support to a different policy.

In addition to housing multiple access decision modules, the ACS contains a software entity called the ACS Module Manager. The Module Manager contains the logic for invoking the appropriate access decision module (i.e., the applicable policy) based upon such parameters as the client or principal making the request, identity of the client during that request (the same client may have different identities from one session to another depending upon which application the client is accessing) as well as the target object and the type of operation invoked.

3.5 Authorization, Policy Rule Specification & Decision Language (APRS-DL) – Constructs and Capabilities

The APRS-DL specifies the authorizations and rules. Rules embed the dictates of the policy that governs the enforcement of access control. Different policies can be specified on different objects. In addition, different policies may govern the access on an individual object depending upon the application client that makes the object invocation request. The APRS-DL should be expressive enough to describe the privilege attributes of the user (e.g., membership in roles/groups, capability lists etc), control attributes of the target object (e.g., Access Control List (ACLs), labels etc) as well as context information (e.g., Session ID, time of access) for well-known policy implementations and authorization management concepts. To illustrate these features, we have chosen a discretionary access control (DAC) policy with a Role based Access Control (RBAC) authorization management concept as an example. We use the acronym DAC-RBAC to refer to this policy. A brief overview of this policy is given in section 3.5.1. Since APRS-DL is a first order predicate logic based language there are two basic constructs for expressing policy information – facts and rules. The facts used within the context of DAC-RBAC are represented using predicates of order two, three or four. The rules are constructed using operations ***&*** (conjunction), ***V*** (disjunction or OR) and ***NOT*** (negation).The facts and rules are discussed in sections 3.5.2 and 3.5.3 respectively.

3.5.1 DAC-RBAC Policy

Discretionary access control (DAC) with Role based access control (RBAC) authorization management concept is a common form of access control used in commercial database management systems. Typically a database application may consist of data objects owned by several users. The owners of objects can use the concept of a role to conveniently group the privileges associated with a specific job function (though the authorization to create the role itself may be vested with a central security administrator or system security officer). Users who require those privileges are then made members of that role. For certain selected users, the owner of a role (usually the creator of the role) can also assign memberships in that role with a special administrative privilege. This in turn will enable the members of that role to grant this role to other users of the system thus effectively propagating the set of privileges associated with that role. In addition, owners of multiple roles (*or* users who are role members with administrative privilege *or* the central system administrator *or* system security officer) can create hierarchies among those roles (by assigning one role to another) or specify constraints on role memberships (e.g., static separation of duty (SSD)). Further, constraints can also be specified on the activation of these roles during a single user login session (e.g., dynamic separation of duty (DSD)).

3.5.2 APRS-DL Facts for DAC-RBAC

3.5.2.1 Context Information: The information regarding the current login session of the user is represented using the predicate *curr_sess(u,sid)* where the parameter *u* stands for userid and *sid* stands for a unique session id generated by the login manager of the environment through which the user interacts with the system.

3.5.2.2 Assigned Role: The role in which the user is assigned membership is captured through the predicate *assigned_role(u,r)* where *u* stands for the user and *r* is one of the roles assigned to the user *u*.

3.5.2.3 Active Role Set: A user may have been assigned many roles but there may be practical limitations on the number of roles that can be active in any one session. The subset of the roles that needs to be available at the start of any user session can either be designated by the administrator or can be chosen by the user himself/herself. This subset (called the Active Role Set) is represented through predicates *active_set(u,r).*

3.5.2.4 Role Hierarchy: In a RBAC based access control scheme, roles are often organized in a hierarchy by the system administrator to simplify the management of roles as well to reflect the responsibilities of users assigned to these roles within the organization. This information is represented using the predicate *child_of (r1,r2)* which denotes that role *r2* is a child of role *r1* (stated in another way – role *r1* is a parent of role *r2*).

3.5.2.5 Role Grants: These specify the permissions granted by object owners or System Administrator or System Security Officer to a role. These facts embody the “need to know” decisions for users belonging to a role (by virtue of the job function the role represents) and do not imply automatic guarantee of access anytime the user belonging to that role invokes the operation specified in them (since access may be constrained by the rules discussed under section 3.5.3). These predicates are of the form *role_grant (r, o, m)* – role *r* has been granted the permission to execute method *m* on object *o*.

3.5.2.6 Invoked Roles: The user session may have many active roles (see 3.5.3.1 below for how active roles are acquired) but there may be constraints that restrict the exercise of all the active roles within the same session. To enforce these constraints the system needs to keep track of all the roles that are already used (invoked) within that session. This is done through the predicate *invoked_role(sid,r).*

3.5.3 APRS-DL Rules for DAC-RBAC

3.5.3.1 Privilege Attributes: In any access control scheme which uses RBAC authorization management concept, a user derives privileges by virtue of being assigned to (made a member in) one or more roles (a grouping construct for collection of privileges for a specific job function). However, as stated in section 3.5.2.3, the resource overheads associated with the management of user session may dictate the need for specifying a subset of these roles as the active role set. Each of these roles in the active role set are then associated with the unique session ID by the session manager after verifying that each of them are indeed one of the roles assigned to the user ID. This association between a member of active role set and the session ID creates the concept of the active role for the session and is obtained through the following rule:

General Form:
active_role(sid,r) $\leftarrow$ *assigned_role(u,r) & active_set(u,r) & curr_sess(u,sid) –*
Role *r* is one of the active roles for the session *sid*, if *sid* is the current session associated with user *u* and role *r* is one of the roles assigned to user *u* and is in the user's active role set. A collection of such active roles constitute the privilege attributes for the user in that session

3.5.3.2 Session Authorizations: These rules specify the authorizations for a particular operation <*o,m*> in a specific session based upon the fact that one of the active roles for that session has the operation in its list of privileges.

General Form:
auth_execute(r,o,m,sid) $\leftarrow$ *role_grant(r,o,m) & active_role(sid,r)*
The role *r* is authorized to invoke method *m* on object *o* in session *sid*, provided the operation <*o,m*> is one of the privileges granted to role *r* and role *r* is one of the active roles in the session.

3.5.3.3 Descendants of Roles: Apart from knowing the immediate children of an active role, it is necessary to know all the roles that are the children of those child roles as well. In other words we need to know all the descendants of an active role since privileges granted to all descendant roles are available to the session's active role. This descendant relationship is captured through a set of following two rules , one of them being recursive.

General Form:
descendant_of(r1,r2) $\leftarrow$ *child_of(r1,r2)*
descendant_of(r1,r2) $\leftarrow$ *child_of(r1,r3) & descendant_of(r3,r2)*

3.5.3.4 Implicit authorizations: These rules specify derived authorizations through role hierarchies. Expressed in another way, these rules denote authorization for operation <*o,m*> obtained through descendant roles of one of the session's active roles.

General Form:
impl_auth_execute(r,o,m,sid) $\leftarrow$
active_role(sid,r) & role_grant(r_d,o,m) & descendant_of(r,r_d) – this rule states that role *r* has the implicit privilege to invoke method *m* on object *o* in session *sid*, if role *r* is

an active role for the session and if there exists a role r_d that has been granted the permission to invoke method *m* on object *o* and r_d is one of the descendants of *r*.

Example:
impl_auth_execute (Acct_Rep, account, debit,SID1018991004) ←
active_role(SID1018991004,Acct_Rep) & role_grant(Teller,account,debit) &
descendant_of(Acct_Rep,Teller)
states that the Acct_Rep role has the implicit authorization to perform the debit operation on the account object in session SID1018991004, if Acct_Rep is one of the active roles for the session and if the Teller role has been granted the permission to invoke the debit operation on account object and Teller is one the descendants of Acct_Rep role.

3.5.3.5 Authorization Annulment Conditions: Even though *session authorizations* and *implicit authorizations* may to some extent guarantee access to the user with requisite privileges for a given operation *<o,m>* ,an organization may need to place some constraints on the usage of the roles (which serve as the medium for transmission of privileges). The enforcement of these constraints may result in annulment of certain authorizations obtained directly through any of the session's active roles or implicitly through descendants of active roles. Hence it is necessary to check for each user requested operation *<o,m>* whether any of these annulment conditions apply.

General Form:
auth_annul (o, m, sid, rct) ← *[auth_execute (r,o,m,sid)* **V** *impl_auth_execute(r,o,m,sid)]*
& invoked_role(sid,r1) & role_constraint (r,r1, rct)

- an authorization annulment condition for operation *<o,m>* in session *sid* due to a role constraint type *rct* has arisen due to the fact that this operation obtained its authorization (directly or implicitly) through a role *r* and this role has a role constraint of type *rct* with another role *r1* that was invoked during the same session. We need this type of rule in situations where there may be two roles available for a user in a session but only one of them can be exercised in that session. An example of this is the dynamic separation of duty (DSD) constraint.

Example:
auth_annul (acct_tran, modify, SID1020991009, DSD) ←
auth_execute (Accountant, acct_tran, modify, SID1020991009) &
invoked_role (SID1020991009, Accts_Mgr) &
role_constraint (Accountant, Accts_Mgr,DSD)

- An authorization annulment condition (due to DSD constraint) has arisen in session SID1020991009 for the operation of modifying an accounting transaction, due to the fact that the user has obtained the authorization for this operation through the Accountant role but has invoked the Accts_Mgr role during this session and the roles Accountant and Accts_Mgr are subject to DSD constraint between them. Very often such annulment conditions are specified for protecting the integrity of some operations. In this example, it may be the case that the Accts_Mgr role is authorized to post the accounting transactions to the general ledger and the intent is prevent the transaction modification operation and the posting operation from taking place in the same user session.

3.5.3.6 Access Decision Rules: These are the final rules used in the accession decision process. The outcome of the evaluation of these rules generates YES / NO flags (ALLOWED / DENIED message) with respect to a particular access request. Access is allowed for a given operation *<o,m>* based upon the presence of either session authorization (predicate *auth_execute(r,o,m,sid)*) or implicit authorization (predicate *impl_auth_execute(r,o,m,sid)*) and absence of authorization annulment conditions (predicate *auth_annul(o,m,sid,rct).*

General Form:
access_allowed (o,m,sid) ←
    [auth_execute (r,o,m,sid) ***V*** *impl_auth_execute(r,o,m,sid)] &*
    NOT *(auth_annul(o,m,sid,rct))*

Thus, we see that the constructs in APRS-DL are capable of processing DAC policy rules using the RBAC authorization management concept. Also by selective combination of predicates *auth_execute* , *impl_auth_execute* and *role_grant* , we can express capability lists associated with a user (an example of a user privilege attribute) as well as an access control list (ACLs) (an example of a control attribute associated with a target object). Also any context information can be incorporated by associating various attributes with the session identifier *sid*.

3.6 APRS-DL Facts/Rules processed in Client Side ACS (ACS/Cl_ORB)

As already stated, the function of client side ACS (ACS/Cl_ORB) is generation of client credentials using access decision modules. The client credentials consists of (a) Context Information and (b) Privilege Attributes. In the DAC-RBAC access control policy environment, we have considered only the session ID (represented by predicate *curr_sess(u,sid)*) under the category of context information and the session's active roles (represented by *active_role(sid,r)*) under the category of privilege attributes. The predicate *curr_sess(u,sid)* is a fact generated by the session manager before the creation of the session after the user is identified and authenticated. The predicate *active_role(sid,r)* is at the head of the rule (given in section 3.5.3.1) whose body contains the predicates *assigned_role(u,r)*, *active_set(u,r)* and *curr_sess(u,sid).* We have already stated the source for getting the fact *curr_sess(u,sid).* The facts for *assigned_role(u,r)* and *active_set(u,r)* are obtained from User Login Information file.

3.7 APRS-DL Facts/Rules processed in target side ACS (ACS/Tr_ORB)

The main function of the target side access computation server (ACS/Tr_ORB) is the generation of ALLOWED/DENIED message in response to every user access request by calling on the appropriate access decision module. For performing this function, the

the truth value of the predicate *access_allowed(o,m,sid)* at the head of the Access Decision Rule (section 3.5.3.6) must be determined. To process this rule, the predicates *auth_execute(r,o,m,sid), impl_auth_execute(r,o,m,sid)* and the set of predicates dealing with annulment conditions must be evaluated. The logical sequence for evaluation of these predicates is described in terms of the following three steps:

(1) Using the list of active roles for the current user session (denoted by predicate *active_role(sid,r)*) in the form of client credentials, the access decision module (within ACS/Tr_ORB) processes the session authorization rule (3.5.3.2) to find a role that will the authorize the requested operation <*o,m*> for the user session *sid* (i.e., find the *r* that will make the predicate *auth_execute(r,o,m,sid)* true). If it is able to find such a role, then it proceeds to process the list of rules relating to annulment conditions (3.5.3.5) (step 3), otherwise it goes to step (2).

(2) The access decision module seeks a role among the descendants of session's active roles that can authorize the user operation <*o,m*>. For this, it evaluates the implicit authorization rule (3.5.3.4). If this results in the access decision module finding an active role which has a descendant that has the permission to perform operation <*o,m*> (i.e., find the *r* that will make the predicate *impl_auth_execute(r,o,m,sid)* true), it proceeds to step (3) to process annulment conditions. On the other hand, if the access decision module does not find a matching rule (i.e., which means that the user operation <*o,m*> is not authorized in any of the session's active roles or its descendant roles), it indicates the failure of access computation process to ACS/Tr_ORB which then generates the access DENIED message and sends it to client side ORB through the target side ORB. The client side ORB in turn communicates this message back to the user who requested the operation <*o,m*>.

(3) Upon successfully finding a role that has the permission to perform operation <*o,m*> (through step (1) or (2)), the access decision module tries to find a rule among all the rules that pertain to annulment conditions (3.5.3.5) that will bind with that role. If all the annulment conditions relating to the operation <*o,m*> using that role evaluate to false, then the access decision module communicates the success of the access computation process to server side ACS(ACS/Tr_ORB) which then generates the access ALLOWED message and sends it to target side ORB. On receiving this message, the target side ORB passes on the user requested operation to the target object.

4. Implementation of MACP

It is clear from the above discussions that the key component of MACP is the Access Computation Server (ACS) which contains multiple access decision modules and a management module. The access decision modules located in client side ACS (ACS/Cl_ORB) will only generate credentials whereas those on the target side ACS (ACS/Tr_ORB) will generate the access ALLOWED/DENIED messages. All access decision modules are defined using APRS-DL constructs. Since APRS-DL is a first order predicate logic based language, the policy attributes and rules definitions in APRS-DL can be implemented in any first order logic based language system like Prolog. There are now translating compilers available for translating prolog code to C++ and Java routines. Using these it is possible to translate the database of authorizations and policy rules expressed in APRS-DL to Java (or C++) interfaces and class definitions. The modules containing these interfaces can then be used as plug-ins to the access computation server. These modules then provide different access control policy support needed for different user requests.

A standardized application programming interface (API) can be defined for the Management Module of the access computation server. This will make the access computation server callable from ORBs offerings from different vendors.

5. Conclusions

Although it is nearly three years since the first CORBASEC security reference model has been published, there are only a few commercial implementations that can claim conformance to CORBASEC specifications. We feel that one of the practical difficulties of developing a security architecture using the CORBASEC framework (which at the same time meets its security goals) is in the area of developing a security service package callable from different ORBs while at the same providing support for multiple security policies. The MACP architecture presented here addresses this problem with respect to access control functionality. The MACP is intended to an architecture whose components can be implemented using existing software development toolkits. The use of a logic based language (APRS-DL) for implementing an access decision module for supporting a DAC-RBAC access control policy was illustrated. The flexibility of the language can be used for defining policy modules associated with other access control schemes like Object-Oriented Domain Type Enforcement (OO-DTE).

Acknowledgement

The author wishes to convey his special thanks to Prof. Sushil Jajodia, Chairman , Department of Information & Software Engineering , George Mason University, Fairfax, VA for providing valuable insights re: the requirements for the authorization and policy rule specification & decision language (APRS-DL).

References

[branstad88] M. Branstad, H. Tajalli, F. Mayer, and D. Dalva. Access mediation in a message passing kernel. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 66--72, Oakland, CA, May 1989.

[CSEC98] CORBASEC – Security Service Specifcation v1.2 (November 1998) by OMG.

[fm93] T.Fine and S.E.Minear. Assuring distributed trusted mach. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 206--218, Oakland, CA, May 1993.

[jd96] D.Jonscher and K. R. Dittrich. Argos - A configurable access control system for interoperable environments. In David~L. Spooner, Steven~A. Demurjian, and John~E. Dobson, editors, *Database Security IX: Status and Prospects*, pages 43--60. Chapman & Hall, London, 1996.

[lunt89] T.F. Lunt. Access control policies for database systems. In C. E. Landwehr, editor, *Database Security II: Status and Prospects*, pages 41--52. North-Holland, Amsterdam, 1989.

[ncsc93] O. S. Saydjari, S. J. Turner, D. E. Peele, J. F. Farrell, P. A. Loscocco, W. Kutz, and G. L. Bock. Synergy: A distributed, microkernel-based security architecture, version 1.0. *Technical report, National Security Agency*, Ft. George G. Meade, MD, November 1993.

[oak97] Sushil Jajodia, Pierangela Samarati, and V.S. Subrahmanian. A Logical Language for Expressing Authorizations. In Proceedings of IEEE Symposium on Security and Privacy, pages 94--107, Oakland, CA, May 1997.

[rbkw91] F.Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. In *ACM Transactions on Database Systems*, 16(1):89--131, March 1991.

[sigmod97] Sushil Jajodia, Pierangela Samarati, V.S. Subrahmanian, and Elisa Bertino. A Unified Framework for Enforcing Multiple Access Control Policies. In *Proceedings of the 1997 ACM International SIGMOD Conference on Management of Data*, Tucson, AZ, 1997.

Implementation of Multiple Access Control Policies within a CORBASEC framework

Ramaswamy Chandramouli (NIST)

Major Contributions

- Identify the fundamental bottleneck in supporting access control policies in distributed systems
- Give an overview of CORBASEC security reference model and select a security model from an access control viewpoint.
- Develop an architecture (with acronym MACP) for defining and enforcing multiple access control policies in the context of the chosen CORBASEC security model

Need for Multiple Access Control Policies

- Protection needs for data in different components of a distributed system are different.

- Different Access Control policies need to be supported

- <u>Main Bottleneck</u>

  - Most Access Control Models & their enforcement mechanisms support only one type of access control policy.

- In other words, Access Enforcement is tied to Access Policy

Need for Multiple Access Control Policies (contd ..)

- Solution for the Bottleneck

  Decouple (1) Policy Definition & Access Computation process from (2) Access Enforcement Mechanism

- We need an Access Computation Engine (ACE) that performs task (1) based on an Access Authorization Model in which we can define multiple access control policies

- ACE can send a message to (2) to allow/deny the access

- Thus enforcement mechanism is not longer policy dependent

CORBASEC - Security Reference Model

- A framework for providing various security functionality in the context of CORBA based distributed systems.

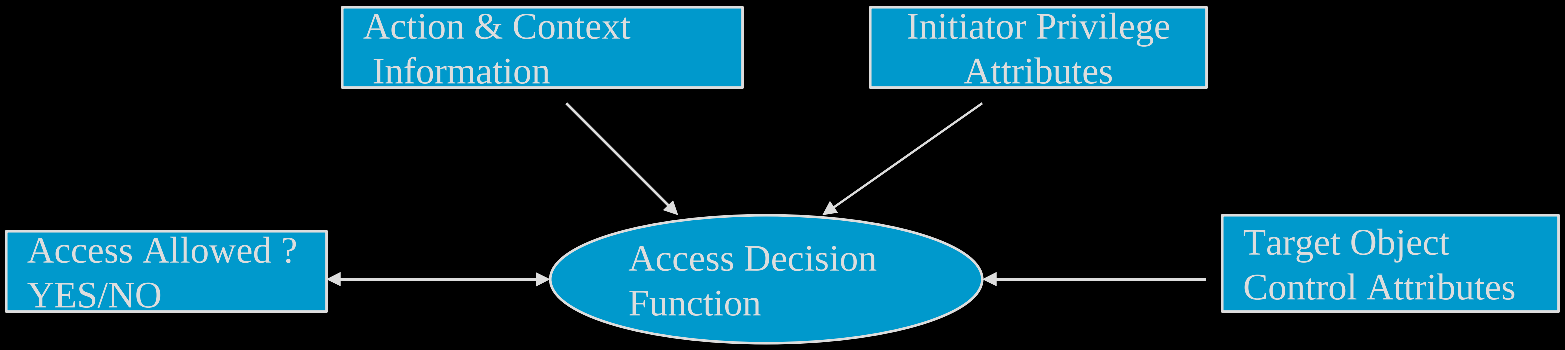
- Security Functionality includes
  - (a) Identification & Authentication
  - (b) Authorization & Access Control
  - (c) Security Audit
  - (d) Security of Communication & Non-Repudiation
  - (e) Security Management

- CORBASEC specifications are technology neutral & provides for freedom of implementation.

SELECTED CORBASEC Security Model (Access Control View)

- Access Control Policy Enforcement Layer
- At the System Layer by the distributed ORB
- Applications need not be security aware

- Access Authorization Model
- Based on abstract Access Decision Functions



Action & Context Information
Initiator Privilege Attributes
Access Allowed ? YES/NO
Access Decision Function
Target Object Control Attributes

Multiple Access Control Policies in a CORBASEC framework

SELECTED CORBASEC Security Model (Access Control View .. contd)

- Packages for providing Access Control Services
- A set of security service replaceability packages
- The packages are used by ORBs both at the client side and target object side

---

- Summary of the selected CORBASEC security Model
- Access Control at System Layer using Security Service Replaceability packages

- These packages provide access control services for the ORBs.

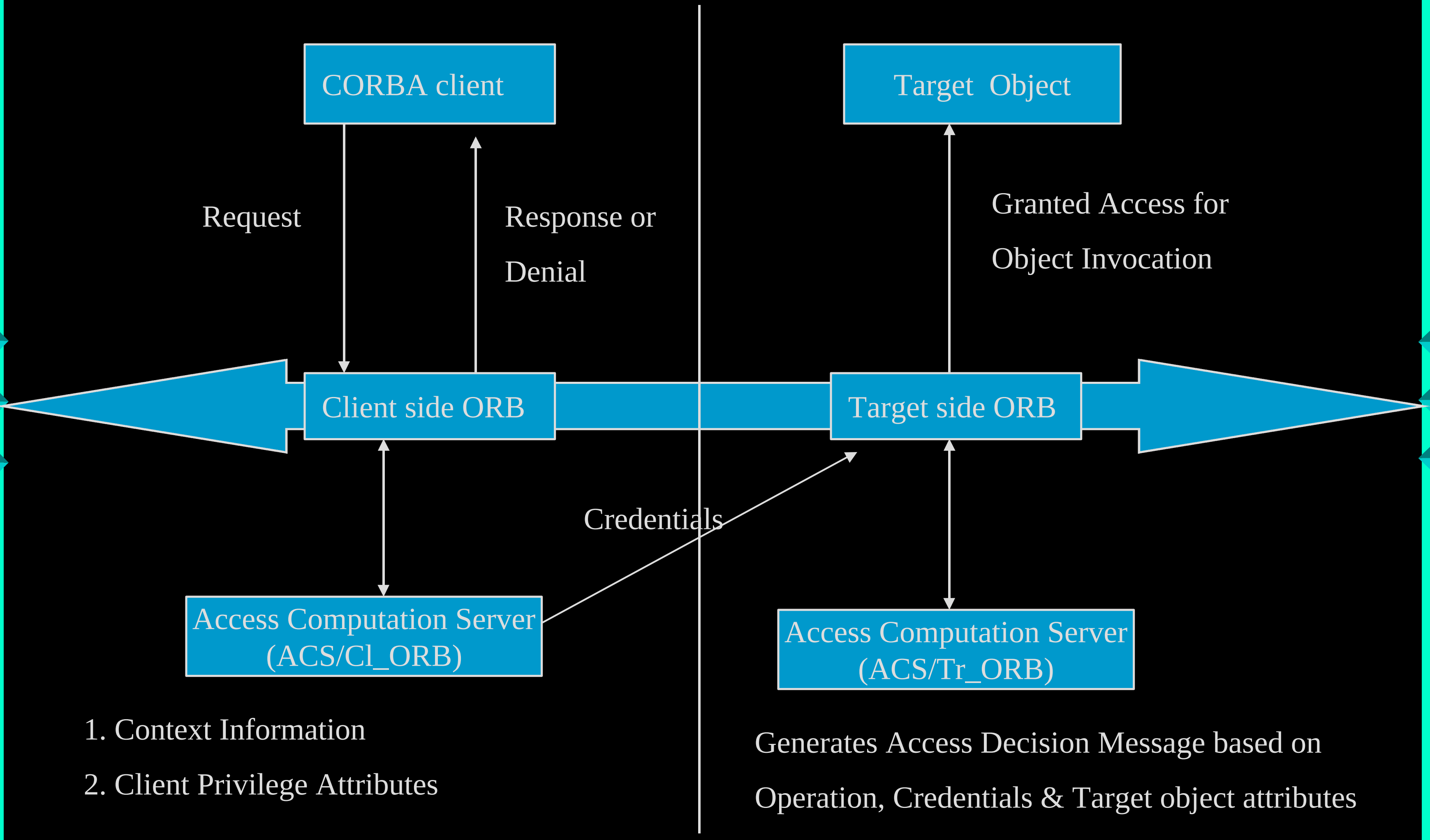
MACP - An Architecture for supporting Multiple Access Control Policies

- The services expected of a Security Service Replaceability package are provided in MACP by an:
  ACCESS COMPUTATION SERVER (ACS)

- The Access Computation Server is present at the client side (ACS/Cl_ORB) as well as target object side (ACS/Tr_ORB)

- <u>ACS Functionality</u>
  (a) provide interfaces for defining access decision modules for different policies using APRS-DL language
  (b) Process client requests & generate access decisions

Components of MACP Architecture


CORBA client
Target Object
Request
Response or
Denial
Granted Access for
Object Invocation
Client side ORB
Target side ORB
Credentials
Access Computation Server
(ACS/Cl_ORB)
Access Computation Server
(ACS/Tr_ORB)
1. Context Information
2. Client Privilege Attributes
Generates Access Decision Message based on
Operation, Credentials & Target object attributes

Multiple Access Control Policies in a CORBASEC framework

Authorization, Policy Rule Specification & Decision Language (APRS-DL)

- Access Decision Modules in Access Computation Server (ACS) for defining various access control policies are implemented using this language
- Logic based - Access Control policies are defined in terms of facts and rules.
- Use of this language for defining a DAC-RBAC policy is illustrated (i.e., facts & rules that need to be represented)

APRS-DL Facts for DAC-RBAC

- Context Information: *curr_sess(u,sid)*
- Assigned Role: *assigned_role(u,r)*
- Active Role set: *active_set(u,r)*
- Role Hierarchy: *child_of(r1,r2)*
- Role Grants: *role_grant(r,o,m)*
- Invoked Roles: *invoked_role(sid,r)*

APRS-DL Rules for DAC-RBAC

- Rule for Generating Privilege Attributes

  Active_role(sid,r) ← *assigned_role(u,r) & active_set(u,r) &*
  curr_sess(u,sid)

- Rules for capturing role descendants

descendant_of(r1,r2) ← *child_of(r1,r2).*
descendant_of(r1,r2) ← *child_of(r1,r3) &*
descendant_of(r3,r2)

APRS-DL Rules for DAC-RBAC (Contd..)

- Rule for Generating Session Authorizations

  auth_execute(r,o,m,sid) ← role_grant(r,o,m) &
  active_role(sid,r)

- Rules for Implicit Authorizations

  impl_auth_execute(r,o,m,sid) ← active_role(sid,r) &
  role_grant(r_d,o,m) &
  descendant_of(r,r_d)

- Rules for Authorization Annulment Conditions

  auth_annul (o, m, sid, rct) ← [auth_execute (r,o,m,sid) **V**
  impl_auth_execute(r,o,m,sid)]
  & invoked_role(r1, sid)) &
  role_constraint (r,r1, rct)

APRS-DL Rules for DAC-RBAC (Contd..)

- Access Decision Rule
- Final rule in the access decision process
- Generates access ALLOWED/DENIED Messages

access_allowed (o,m,sid) ← *[auth_execute (r,o,m,sid)* **V**
impl_auth_execute(r,o,m,sid)] &
NOT *(auth_annul(o,m,sid,rct))*

MACP - Summary of Features

- The key component of MACP is the Access Computation Server (ACS).
- The ACS consists of two categories of Modules
  - (1) Multiple access decision modules for supporting different access control policies
  - (2) A management module that determines the appropriate access decision module (thereby the applicable policy) to use for each user request

- APRS-DL constructs can be implemented using a first order predicate logic based language & converted to other languages (like Java & C++) using translating compilers.

Conclusions

- Very few commercial implementations for CORBASEC specification

- The Main problem in the development of a security architecture which is CORBASEC compliant is in developing a security service package that:
  - (a) is callable from different ORBs
  - (b) can support multiple security policies

- MACP addresses this problem with respect to Access Control functionality.