Object Security Attributes: Enabling Application-specific Access Control in Middleware

Konstantin Beznosov
Quadrasis
Hitachi Computer Products (America), Inc., Waltham, MA
konstantin.beznosov@quadrasis.com

Abstract

This paper makes two main contributions towards establishing support for application-specific factors in middleware security mechanisms. First, it develops a simple classification framework for reasoning about the architecture of the security mechanisms in distributed applications that follow the decision-enforcement paradigm of the reference monitor. It uses the framework to show that the existing solutions lack satisfying trade-offs for a wide range of those applications that require application-specific factors to be used in security decisions while mediating access requests.

Second, by introducing attribute function in addition to decision and enforcement ones, it proposes a novel scheme for clean separation among suppliers of middleware security, security decision logic, and application-logic, while supporting application-specific protection policies. To illustrate the scheme on a concrete example, we describe its mapping into CORBA Security.

Keywords
Middleware Security, Access Control, Authorization, CORBA Security, SDMM.

1 INTRODUCTION

The employment of application-specific factors in security decisions is not new. Target ADI in OSI access control framework is an example [1]. Most today commercial distributed application security systems [2-5] provide various levels of support for the factors. What missing is a systematic inclusion of the support for these factors in the architecture of access control and other security mechanisms of middleware systems.

The void results in the lack of adequate architectural provisions in middleware security leading to ad-hoc solutions. Although some efforts are under way to develop support for object security metadata in authorization policies [6], our analysis did not reveal any notable work done in the research community on support for application-specific factors in the security mechanisms of distributed applications. Systematic support for application-specific factors is necessary but it is missing in the architecture of distributed applications.

In this paper, we propose a schema for systematic support of application-specific factors in security mechanisms for distributed applications. To facilitate analysis of the existing solutions for supporting such factors as well as to describe the proposed solution, we develop a framework for reasoning about those security mechanisms in distributed applications that follow decision-enforcement paradigm (such as access control, audit and quality of message protection). The framework allows classifying all solutions into four major schemes.

We use the framework to introduce our solution, which has two main components: generic representation of application-specific security-related factors in the form of object security attributes, and additional function for retrieving them at the time of access request mediation by the security sub-system. Being conceptually simple, the approach enables the use of application-specific factors in security policy decisions without coupling evaluation engines and target objects while keeping underlying middleware security application-neutral. Therefore, the security decision function can be provided by a third-party, while the enforcement function stays in the middleware freeing the application owner from implementing either. To substantiate relatively-abstract explanation of the proposed solution, we show its translation into a concrete architecture of CORBA Security.

The rest of the paper is organized as follows. The problem to be addressed is stated in the next section. The framework is introduced and the available solutions are discussed in Section 3. Section 4 presents our solution in generic form. Its concrete application to CORBA Security is discussed in Section 5. Discussion is provided in Section 6. Conclusions are drawn in Section 7. Acknowledgments can be found in Section 8. Section 9 contains references.

2 PROBLEM MOTIVATION

The problem raises because of the conflict of the following forces. On the one hand, a capable implementation of middleware security is usually a complex and expensive piece of machinery, and is somewhat similar in its generality to operating system security. Due to its critical nature, it needs to be carefully designed, implemented, tested, assured, and tuned for performance and scalability. Thus it is essential for the producers of middleware security to avoid changes to their products and yet apply them to diverse application domains.

On the other hand, there is strong and natural interest among numerous owners of distributed applications in making security decisions, mainly authorization ones, based on factors specific to the applications and organizational workflow, which is some times referred as object security metadata, as in [6].

2.1 WHAT ARE APPLICATION-SPECIFIC FACTORS

Unlike a resource security attribute, an application-specific factor is a certain characteristic or property of an application's resource produced, modified, and processed in the course of normal application execution and not for the sole purpose of a security policy decision. In OO middleware and other distributed systems, application objects are such resources. A remote analogy could be resource access control decision information (ADI), defined as a description of the resource's security-relevant properties, in ISO access control framework [1]. The difference between ADI and application-specific factors is that the former could be administered and utilized for the sole purpose of making access control decisions.

2.2 EXAMPLE

For the sake of illustrating the needs of user organizations, consider the following real-life examples from banking and telecommunication domains. Several people could be associated with each bank account each having different rights. For example, primary holder could do everything, including deleting the account, whereas secondary holders, depending on the loyalty of the primary holder, could have different levels of limited access, such as withdrawing limited amounts, reviewing, etc. All others can only deposit to the account. Implementation of such policies requires dynamic evaluation of the relationship between the accessing subject and an account. List of account holders and their "rank" (i.e. primary, secondary, etc.) are such application-specific factors.

Security policies that U.S. long distance telephone carriers need to enforce depend on the state (e.g. Florida, Pennsylvania) a particular account is in. At the same time, accounts change phone numbers (and therefore possibly state) due to the relocations of their owners. Appearing to be a little and relatively infrequently occurring task, manually re-associating an account object with the corresponding state's policy becomes a resource consuming operation for carriers with millions of subscribers (relocating on average every 5 years over 5,000 per day for 10^6 subscribers). Instead, the policy could be determined using the first 6 digits of the phone number, which becomes one of the application-specific factors to be used in security policy decisions.

2.3 OBJECTIVE

Also advocated in by others ([6]) and identified as one of the input types for access decision function in [1], this demand for the use of application-specific information in authorization and other security decisions has clear rationale. The more application or workflow information is used in security decisions the better is the integration between security and application administration, which leads to low administration costs and fewer errors of application and security administrators. Even more importantly, enforcement of application-specific policies in the middleware frees developers from coding such policies in their applications thus making systems less complex, quicker to build, and easier to evolve. These two factors mean significant long-term savings for application developers and owners.

At the same time, a number of enterprise-scale authorization products first appeared on the market of web servers security, and then expanded into application servers. These systems are perfect candidates for providing authorization services to distributed applications and the underlying middleware security. However, as we will show it in the next section, there are technical obstacles in integrating them with either the former or the latter.

Naturally a question arises: is there a way to keep middleware security services generic and yet allow for enforcement of security policies specific to different application domains, possibly with the use of enterprise authorization systems?

3 AVAILABLE SOLUTIONS

For the purpose of analyzing solutions to the problem, we differentiate all approaches to security policy decision and enforcement in distributed systems by two factors. They are the nature of policy decision and enforcement functions. Roughly, each of these functions can be provided either by the distributed application itself, or by the security subsystem of the underlying distribution infrastructure, i.e. middleware security. Using acronyms defined in Table 1, we have four

	Decision Function	**Enforcement Function**
Middleware	MD	ME
Application	AD	AE

Table 1: Acronyms for different locations of security policy decision and enforcement

possible combinations of decision and enforcement: MDME, ADAE, MDAE, and ADME. In the following subsections, we explain each scheme and use this classification to argue that the available solutions do not provide desirable trade-offs.

3.1 MDME -- EVERYTHING IS DONE BY MIDDLEWARE

The first case is the most obvious, when both functions of security decision and its enforcement are provided by the middleware security, as shown in Figure 1. This is what practical

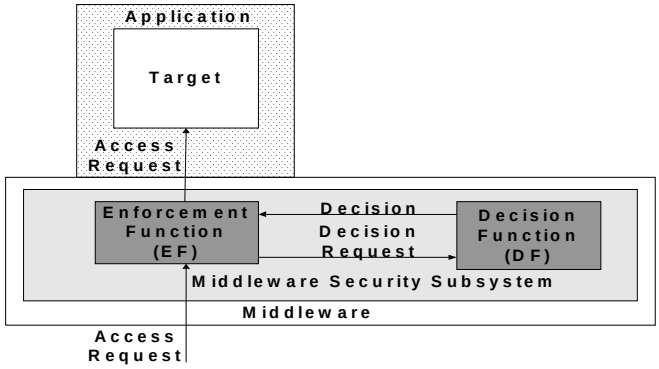

Figure 1: MDME Schema

middleware security systems implement. Being generic, both decision and enforcement functions come with the security subsystem. Applications are security-unaware, and therefore are easier to design, develop, test, deploy and support. In addition, since the reference monitor is not "spilled" over into the application layer, assurance efforts are limited to the middleware layer and those below. This is why MDME schema is considered to be the best for the purpose of enterprise security integration. However, with this approach no application-specific factors can be used for security policy decisions. Consequently, MDME schema is of no use for addressing the stated problem.

3.2 ADME -- APPLICATION DECISIONS ARE ENFORCED IN MIDDLEWARE

A more capable way is to externalize application-specific security logic into a separate service or module and make the middleware security subsystem to obtain policy decisions from it. This ADME schema is illustrated in Figure 2.

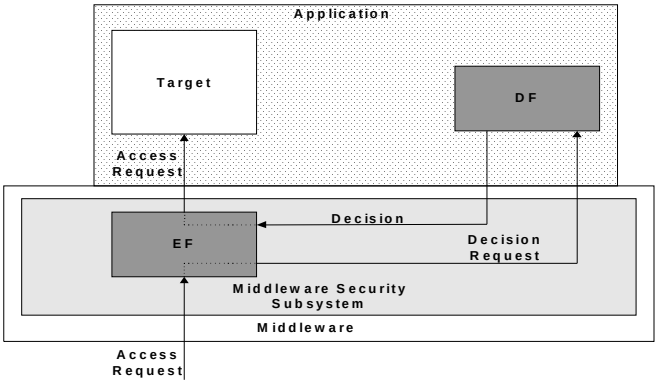

Figure 2: ADME Schema

The schema has been realized in a number of middleware architectures. CORBA Security [7] has replaceable AccessDecision and other interfaces. Java authentication and authorization service (JAAS) [8, 9], which recently became a part of J2SDK v1.4, has replaceable interface Policy that serves authorization decisions.

Although appearing to be versatile, ADME schema has two major drawbacks. Firstly, even being application-specific, decision function (DF) still has the same generic interface for the enforcement function (EF) to query it because the latter remains generic. The interface protocols are not capable of communicating application-specific information between EF and DF. For instance, operation access_allowed() in CORBA's replaceable AccessDecision interface takes subject's credentials, reference to the target object (just "target" for short), target type, and operation on it. Java's Policy::getPermissions() accepts, as input parameters, information representing original location of the code (where the code came from) and the public key(s) of its signer. Microsoft's .NET security model, although not well documented at this time, appears to have security architecture similar to Java in this regard. Clearly, neither of them provide for application-specific factors to be communicated from enforcement to decision point.

And, even if the DF interface supported communication of application-specific factors, the means of retrieving such factors in the EF are not defined. Thus, a custom implementation of DF would have to use a back door to go back to the target object (or a data repository) and retrieve application-specific factors from there, as shown in Figure 3.

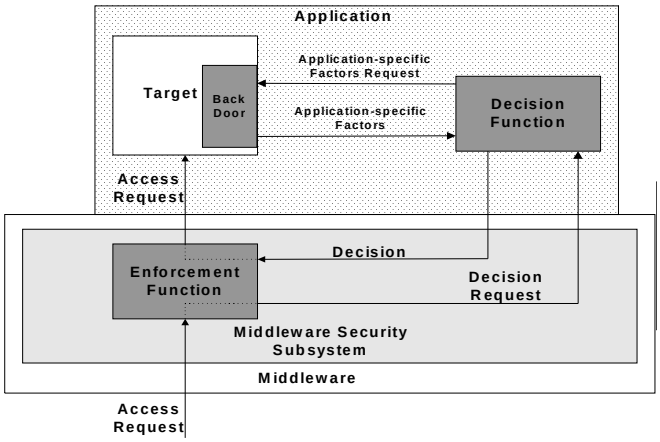

Figure 3: Back Doors to Target in ADME Schema

If a back-door is provided by a target object, the main drawback of the technique is due to the performance hit. Specifically, in some applications, target objects could be very expensive to restore their state and make them ready to serve requests (including those coming through back doors). Consider applications that use secondary or even ternary storage for storing object state between sessions. It could be prohibitively expensive to restore an object, obtain some data through its back door, and make authorization decision just to find out that the access has been denied. Performing expensive reincarnation of target objects for making security decisions also creates a vulnerability for denial of service attacks.

Despite of DF having a potential to be specific to the application domain, ADME schema allows enforcement of only those run-time pre-requisites, such as (dynamic) conditions in [10, 11] and [6], obligations in XACML [12] and provisions in [13], that are non-specific to application domains (such as CPU load). This limitation is due to EF being part of the generic middleware layer.

Secondly, being all-or-nothing solution, the use of application-specific decision function forces the new logic to re-implement authorization decisions completely, which is prohibitively complex and difficult to do correctly for distributed large scale systems, thus rendering this approach unrealistic for most user organizations with the needs in application-specific authorization and other security logic. End-user organizations are just do not want to be in the business of implementing authorization (and other security policy) evaluation engines, which is required if application-specific factor were to be used in security policy evaluation.

3.3 ADAE -- SECURITY-AWARE APPLICATIONS

ADAE schema is more flexible then the previous two. The idea is to let an application-provided EF to call the DF (also provided by the application) thus obviating the problem of obtaining application-specific factors by a generic EF. The schema is illustrated in Figure 4. Although being generic, the

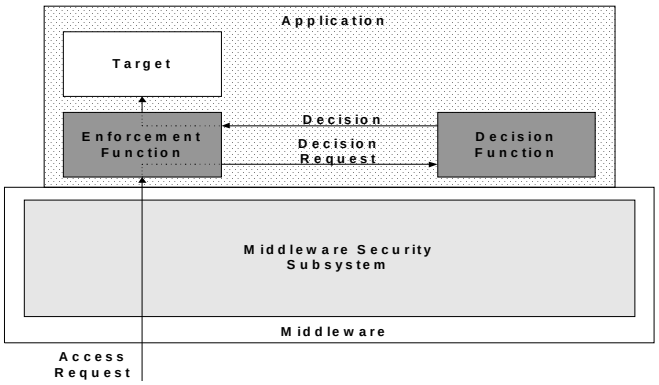

Figure 4: ADAE Schema: General Case

protocol of supplying necessary information to a DF and retrieving decisions from it can be used for communicating application-specific factors.

The general case (Figure 4) of this schema, when EF is external to targets, although being employed in some research systems [14-17], is not known to be popular in real-life solutions. We believe, this is because it requires a proxy object implementing EF to "wrap" each target, and it does not allow enforcement of fine-grain policies because EF is outside of the target.

ADAE With Target Implementing EF

However, a particular case of ADAE, when EF is implemented in the target, as shown in Figure 5, is widely used. It is

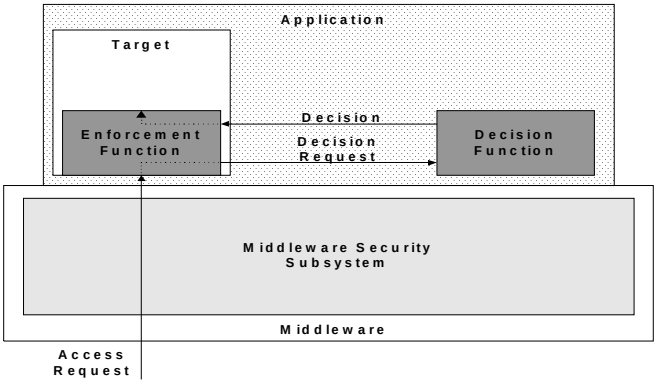

Figure 5: ADAE Schema: target implements EF

popular in distributed application systems constructed on top of limited middleware security technologies (for instance, those lacking access control enforcement, such as Kerberos [18] and SESAME [19]) or implementations. Another reason for employing ADAE schema with EF implemented in targets is the capability to enforce fine-grain security policies, and the ease of obtaining application-specific factors because of the collocation of the business and enforcement functions.

One example of the approach is Resource Access Decision (RAD) architecture [20, 21], where a resource name, composed of a name-value pair list, can be used for encoding application-specific factors. In our example with shopping cart application customer ID number, encoded in the name of the resource in question, can be now used during policy evaluation.[1]

Other examples of middleware security systems where enforcement function is implemented by an application system are Praesidium from HP [22], Adage [23], GAA API [10, 11, 24], and Access Control Unit in [6]. As in RAD, these solutions feature an authorization function invoked by an application for obtaining access control decisions, which are expected to be enforced by an application.

Unfortunately, this active role of the target in composing queries to DF and enforcing decisions results in a number of disadvantages. The most salient one is the necessity for security-related code to be mixed with business logic making targets security-aware. As we argue in [25], this security awareness by target objects makes them more complex and prone to security vulnerabilities. It also makes more difficult to perform security assurance, and forces application developers to be experts in security programming despite of externalizing security decision logic.

Being the best and sometimes the only viable solution for particular cases with complex, application-specific, or fine-grain security policies, programming EF inside of target objects is still a sub-optimal approach for those application domains where a combination of a general purpose security decision function with application-specific attributes could suffice.

3.4 MDAE SCHEMA

We are not aware of any solutions that employ MDAE schema. This is not surprising since application-executed enforcement of security decisions made by the middleware security subsystem does not seem to bring any advantage over any other scheme, while having all the disadvantages and limitations of MDME schema and some of ADAE.

3.5 REAL LIFE HYBRIDS

Several commercial solutions for securing web and middleware applications [2-5, 26] implement an authorization server. It can be queried by either an application itself (ASP or JSP in a web server, bean in EJB container, or CORBA/COM application object), or a middleware-specific enforcement function (web server filter, EJB container, or CORBA security interceptor). Although widely employed in large organizations, neither of these usages introduces any novel solution to

1. Clearly, this requires appropriate programming of the decision function.

the stated problem. The former is an instance of ADAE and the latter -- ADME schemes.

4 GENERIC SOLUTION -- ADME/AF SCHEMA

We introduce a new approach, which enjoys the advantages of ADME schema and yet enables the use of application-specific factors in security policy decisions without coupling evaluation engines and target objects. Therefore, DF can be provided by a third-party company, including an authorization product vendor, while EF stays in the middleware freeing the application owner from implementing either.

As we discussed in Section 3.2, though allowing security decision logic to be application-specific, original ADME schema suffers from the lack of the means (a) to communicate application-specific factors to DF, and, most importantly, (b) to obtain them given the target object in question.

We address the first, simpler, problem by introducing generic representation for application-specific factors. These, as we refer to them, object security attributes (OSA) could be expressed in a number of formats varying in complexity from name-value pairs to arbitrary XML-based structures. The semantic interpretation of an OSA is completely up to the processing entity -- DF. In our example with a telecommunication carrier, for each account object there could be an OSA "holding" the current phone number of the account. More than one OSA can be associated with a target object, comprising a collection of OSAs.

We resolve the second issue, obtaining OSAs for the corresponding target object, by introducing additional function in the ADME schema -- attribute retrieving function, or just attribute function (AF) for short, as shown in Figure 6. This

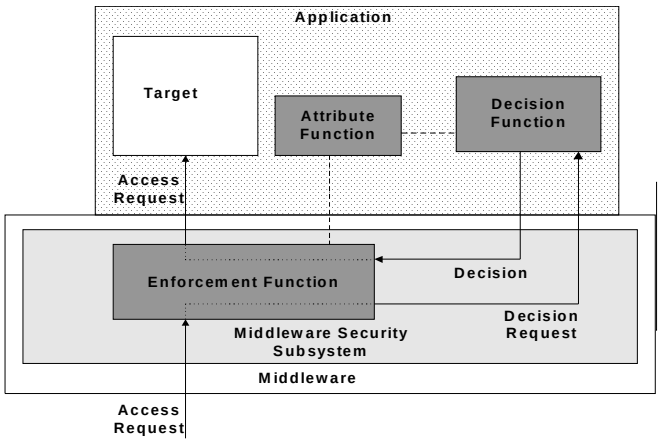

Figure 6: Attribute Function in ADME Schema

function has simple syntax: it accepts (middleware-specific) data necessary for identifying the state of the target object and returns a set of OSAs for that object. The target object state is necessary for retrieving such object metadata as its OSAs.

Since OSA semantics is very specific to the application being protected, AF is provided by the application and not by the middleware or security layers.

What function obtains OSAs via AF is very dependent on the particular implementation of the approach. In some circumstances, EF could be in better position to make an invocation to AF. In others -- DF. Even more, some implementations could make EF and DF to perform this step together. For example, in Section 5, we show how CORBA-specific realization of ADME/AF schema splits AF into two objects, one of which is invoked by EF and the other by DF.

There is a number of advantages, if AF is invoked by EF. First, data, necessary for identifying the target state that EF has at its disposal at the invocation point is very specific to the middleware technology and the type of the particular object adapter that hosts the target. Therefor, for a DF to obtain OSAs from AF, EF would have to pass such data to DF. Second, since DF is usually a COTS, which serves authorization decisions in other schemes, such as ADAE, and for different middleware systems, decision interface to it is too generic to support target state data. Third, there are could be more than one DF (one for each type of security policy, i.e. authorization, audit, quality of protection, non-repudiation, etc.) invoked at every access to a target. It seems beneficial to minimize the number of potentially expensive invocations on AF by obtaining OSAs once per access request.

On the other hand, postponing invocation of AF allows for lazy strategy, when OSAs are retrieved only if some DF is going to use them. More over, a DF could need only particular OSA(s). Retrieving only needed OSA(s) is simple to implement when invocation to AF is done by DF.

This introduction of OSAs as a way to represent target metadata related to security decisions in distributed applications, and a function for retrieving them, AF, enables security decisions to be application-specific while keeping EF in the middleware security layer and using COTS authorization (and other security policy decision functions) systems without modifications. We refer to this approach as ADME/AF schema.

5 APPLICATION TO CORBA SECURITY

So far, we stated the problem, analyzed available solutions, and presented our approach in a form independent of any particular middleware technology. Even more, we generalized the solution so that it is applicable not only to the access control but also to other security functions that can be decomposed into decision and enforcement phases on each access request. Now, we are going to consider particular middleware security technology -- CORBA security -- and show how the general solution applies to it.

5.1 CORBA ARCHITECTURE

This section provides a description, adopted from [27, 28], of CORBA ORB architecture. The architecture consists of several primary components illustrated in Figure 7 and described below.

Target object (or just object for short) -- a CORBA programming entity that consists of an adapter-specific identity, an interface, and an implementation, which is known as a Ser-

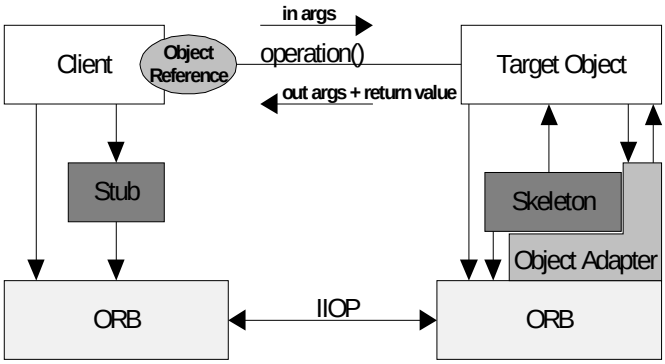

Figure 7: CORBA ORB Architecture

vant. Servant is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.

Client -- the program entity that invokes an operation on an object implementation using **object reference** (OR). Accessing the services of a remote object should be transparent to the caller. Ideally, it should be as simple as calling a method on an object, i.e., obj->op(args). The remaining components in Figure 7 help to support this level of transparency.

Object Request Broker (**ORB**) provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the client and target ORBs are responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller. An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB, which provides various helper functions.

CORBA IDL **stubs** and **skeletons** serve as the "glue" between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.

Object Adapter (OA) assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects). The ORB and the OA cooperate to allow client applications to invoke requests on CORBA objects and ensure that each valid CORBA object is mapped to a servant. In addition, the ORB and the OA cooperate to transparently locate and invoke the proper servants given the addressing information stored in CORBA object references.

The primary type of OA used in today CORBA applications is portable object adapter (POA) [29]. A server application can have multiple POAs nested within it. An application might want to create multiple POAs to support different kinds of CORBA objects and/or different kinds of servant styles. For example, the application might have two POAs, one that supports transient objects and one that supports persistent objects.

A nested POA can be created by invoking a factory operation on another POA. All servers have at least one POA called the Root POA. To create a POA nested under the Root POA, the application invokes the create POA operation on the Root POA. The object reference for the Root POA is available from the ORB. The characteristics of each POA other than the Root POA are controlled at POA creation time using different POA policies.

5.2 RUN-TIME CORBA SECURITY

CORBA Security service (CS) [7] defines interfaces to a collection of objects for enforcing a range of security policies using diverse security mechanisms. It provides abstraction from an underlying security technology so that CORBA-based applications could be independent from the particular security infrastructure provided by user enterprise computing environment. Due to its general nature, CS is not tailored to any particular access control model. Instead, it defines a general mechanism which is supposed to be adequate for the majority of cases and could be configured to support various access control models. CS model comprises the following functionalities visible to application developers and security administrators: identification and authentication, authorization and access control, auditing, integrity and confidentiality protection, authentication of clients and target objects, optional non-repudiation, administration of security policies and related information.

One of the objectives of CS is to be totally unobtrusive to application developers. Security-unaware target objects should be able to run securely on a secure ORB without any active involvement on their site. In the meantime, it must be possible for security-aware objects to exercise stricter security policies than the ones enforced by CS. In CS model, all object invocations are mediated by the appropriate security functions in order to enforce various security policies such as access control. Those functions are part of CS and are tightly integrated with the ORB and the corresponding OAs.

Security policies are enforced completely outside of an application system at the ORB level. Everything, including obtaining information necessary for making policy decisions (such as access control), is done before the method invocation is dispatched to the target object. As Figure 8 shows, policy enforcement code is executed inside of CORBA Security enforcement sub-system, when a message from client application to a target object is passed through the ORB. Executed at

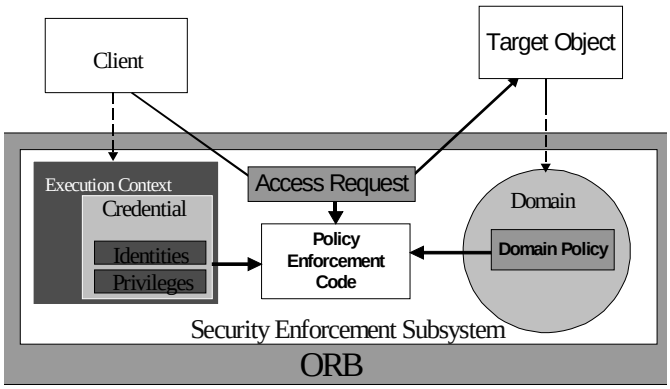

Figure 8: Enforcement of Policies in CORBA Security

the client ORB as well as at the target ORB, the enforcement code uses the following three sources of information for making policy decisions to enforce:

- The policy of the domain(s) to which the target belongs.
- The information from credentials of the client. In case of access control policy enforcement, these are client privilege attributes (such as access identity, group membership, roles and clearance). Where as for audit policy enforcement security attribute of type AuditId is used.
- The access request itself.

Although, a CORBA security sub-systems can be, and usually is, integrated with the ORB and OAs using proprietary means, for the sake of simplicity, we conceptualize its enforcement function as a security interceptor.

5.3 ATTRIBUTE RETRIEVAL FUNCTION

In our solution to the domain of CORBA Security, the closest analogy to AF is interface **AttributeRetriever**.[1] This interface provides operations for retrieving OSAs, as shown in Figure 9. Having its operations implicitly tied into particular

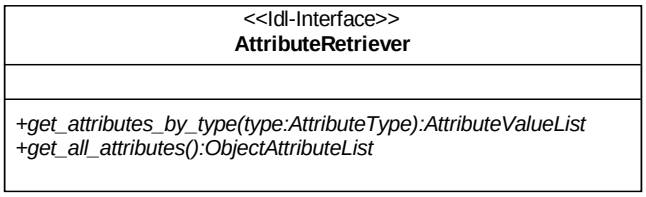

Figure 9: AttributeRetriever Interface

target object, the interface constitutes only part of AF, which is invoked by DF. However, there is another part of AF invoked by EF. This is due to the problem of invoking adapter-specific AF by adapter-neutral DF. AF has to be specific to the type of object adapter because, as in most middleware technologies, the notion of object universal identity is not well developed in CORBA architecture.[2] But object specific identity is sufficiently strong in the context of a particular object adapter, that is, the adapter has sufficient amount of information in order to dispatch an access request to the right object servant, which is responsible for processing application requests for the object in question. On the other hand, it is highly undesirable to have decision functions to be specific to the adapter type.

This is why additional level of indirection via interface **Manager**, the other part of AF, has been introduced. Adapter-specific derivatives of Manager locate AttributeRetriever objects and return them to the EF, which is implemented in the form of a security interceptor. Being adapter-specific, such an interceptor takes control over the access requests, as well as obtains and enforces policy decisions. Before a security interceptor invokes DFs, it obtains a local reference to the correct AttributeRetriever object from the corresponding Manager, as shown in Figure 10, and places it on **Current** object that serves as a thread-specific placeholder for OSA-related information. Later, when the interceptor calls AccessDecision and other objects, their implementations can obtain OSAs from AttributeRetriever via its OR available off Current object. This lazy strategy of making AttributeRetriever available for later queries by DFs allows retrieval of OSAs only if it is necessary. If OSAs are needed more than once per access request, an AttributeRetriever could cache results of time expensive retrieval operation.

5.4 REGISTERING AND DISCOVERING MANAGERS

Another issue is about the means for an application to register and for the interceptor to obtain a local reference to a Manager. Along with this, additional question needed a resolution, specifically what should be the scope of Manager, i.e. should one Manager serve AttributeRetrievers for all objects in a given application. And, if not then how a security interceptor should determine what Manager serves a given object? We saw certain benefit in making the solution flexible and support existence of several Managers for each application. This flexibility allows different implementations of AttributeRetrievers in one application. Moreover, since adapter-specific derivatives of Manager have to be used, the limitation to only one Manager instance would prevent different object adapters from co-existing in the same application.

However, it turned out to be hard to find a way to share multiple Managers among process collocated objects. The essence of the problem is the lack of a placement for the information associating an object with a Manager. An object adapter appeared to be the only appropriate place to store such information with good chances to retrieve it efficiently at the time of mediating an access request. The associating information is stored in the form of **ManagerPolicy**. This lightweight object holds a reference to the Manager instance. The use of

1. Names of all interfaces in this section are relative to ObjectSecurityAttribute IDL module, unless otherwise noted.

2. See [30] for detailed discussion on the shortcomings of object identity in middleware security.

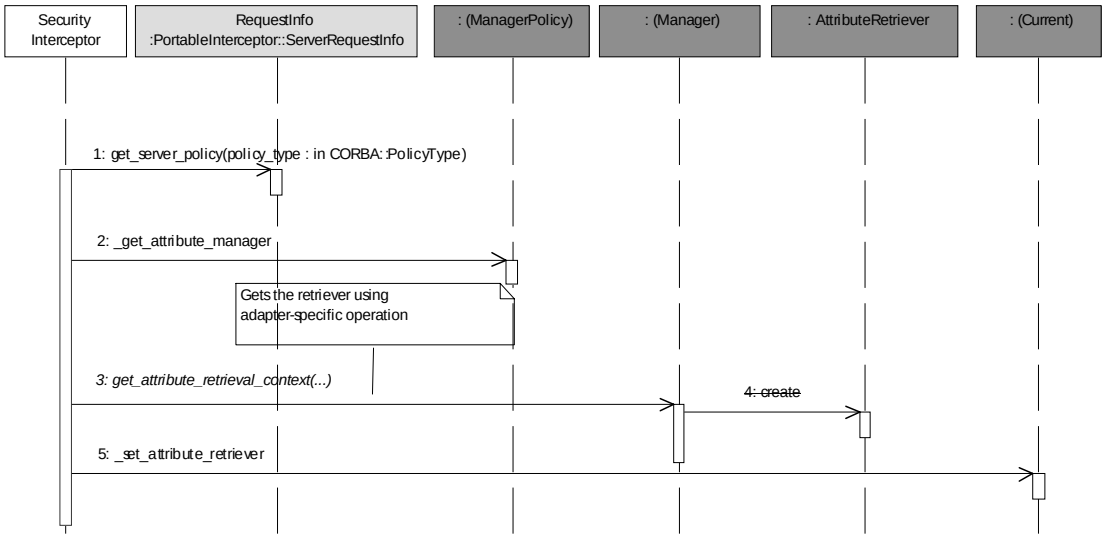

Figure 10: CORBA Security Interceptor Prepares AttributeRetriever to be used by DFs

ManagerPolicy follows the design philosophy exercised in the design of POA, where configuration of a particular POA instance is encoded in the form of POA policies "attached" to the adapter at the time of its creation.

The relationships between target objects, Managers, ManagerPolicies and object adapters is illustrated in Figure 11.

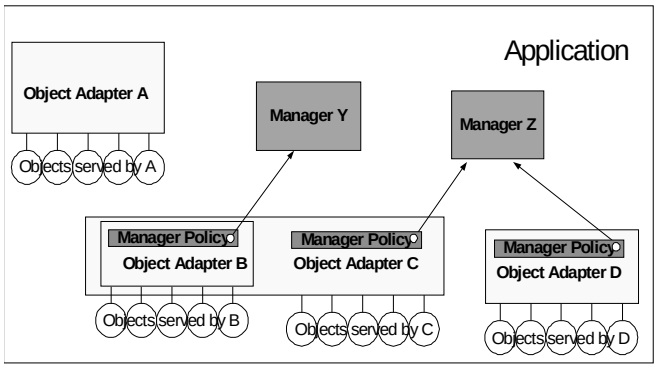

Figure 11: Relationships Among Objects, Object Adapters, OSA Managers, and ManagerPolicies

Access requests for any given object are originally pre-processed and then dispatched by the corresponding object adapter (OA) to the object's servant. OAs can constitute hierarchies with one root, unlimited child and corresponding parent adapters. For each OA there could be no more than one Manager that serves OSAs for all objects under the adapter. That Manager, is said, serves the OA. The same Manager can serve more that one OA, as in the case of object adapters C and D and Manager Z in Figure 11.

Each OA could contain an instance of ManagerPolicy, which holds an object reference to the Manager serving the adapter. If an application provides a Manager for the given OA it sets ManagerPolicy (containing a reference to the Manager) on the adapter using mechanisms specific to the adapter type. For example, such a policy (among others) is passed to the adapter's parent at the child's creation time in the case of POA. If no ManagerPolicy is set on an OA (as for adapter A in Figure 11), then no Manager serves this adapter, which is equivalent to the lack of OSAs associated with the adapter's objects.

The diagram in Figure 12 depicts a sequence of invocations that an application performs for registering its Manager implementation with an OA. After creating an implementation of adapter-specific Manager sub-interface (invocation 1 in the sequence diagram) the application obtains a reference to the Current from the ORB by invoking ORB::resolve_initial_references() with argument "ObjectSecurityAttributeCurrent" (invocation 2). Then, the application uses a factory operation on the Current interface, to which it supplies a valid OR for the Manager, for creating an instance of a ManagerPolicy (invocation 3). While processing the invocation, Current creates an instance of ManagerPolicy and returns it to the caller.

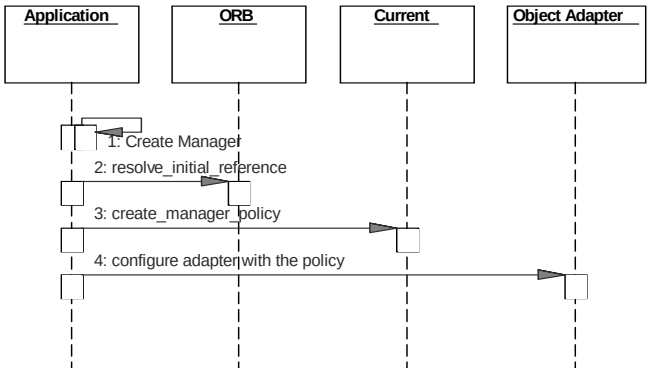

Figure 12: Sequence of Steps Done by an Application for Registering a Manager

The last step (invocation 4) is to "hand" the ManagerPolicy to the OA using the OA-specific mechanisms. For example with POA, an instance of ManagerPolicy is inserted, with other CORBA::Policy objects, in the list of policies provided to the parent POA as an argument of factory operation PortableServer::POA::create_POA() for creating the POA that will serve the same objects as the Manager referred in the ManagerPolicy will.

We summarize the discussion of the CORBA-based implementation of ADME/AF schema with a UML model of the main data types depicted in Figure 13. Being a base interface

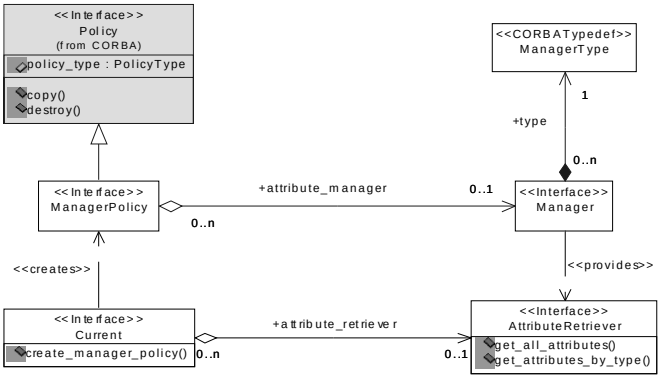

Figure 13: UML Class Diagram for ADME/AF Schema in CORBA

for adapter-specific sub-interfaces, Manager defines only type attribute for indicating the adapter type (e.g. POA) which the sub-interface supports.

Detailed architecture of the solution in the realm of CORBA Security is provided in [31]. The work presented in this section became a CORBA standard adopted by the Object Management Group, and it is currently on finalization track.

6 DISCUSSION

ADME/AF schema is not ideal and does not work for all cases. Being derived from ADME, it would not work for applications that require fine grain protection because its granularity is not finer than methods exposed by a target object. It does not support application-specific pre-requisites either.

Further, if a DF has to call an AF, then its implementation becomes specific to AF interface, yet additional DF wrapping could help. Some middleware technologies, such as COM+ [32], do not provide mechanisms for implementing ADME schemes, which makes applicability of this approach limited. Those rare applications in which security policies are very hard to express using application-specific factors fall out of the applicability scope too.

While having these limitations and disadvantages, the approach gives all of the flexibility of embedding decision functions in the target without requiring the target to be security aware, for those applications whose security policies have moderate granularity requirements, can be expressed using OSAs, and need only decisions enforceable by generic EF.

Our solution enables the process of implementing secure distributed applications to be cleanly separated among

- middleware security suppliers, who implement EF,
- security logic suppliers, who implement DF,
- application suppliers, who implement target objects, and possibly AF, and
- application owners who, having possibly AF implemented, configure EF, AF, and DF to work together and enforce application-specific protection.

While we deliberately limited the proposed approach to the domain of distributed applications, it will be interesting and useful to investigate its applicability and utility for operating systems, as an example.

7 CONCLUSIONS

In this paper, we stated the problem of supporting application-specific factors in security mechanisms in the field of distributed application systems. In order to address the problem, we first created a framework for reasoning about those security mechanisms in distributed applications that follow decision-enforcement paradigm. Using the framework, we showed that all cases can be partitioned into four schemes depending on whether middleware security or application provides decision and/or enforcement functions. Being most promising solution for the stated problem, ADME schema lacks, however, the means of obtaining and communicating application-specific factors to DF. In the described solution, we introduced the notion of an object security attribute (OSA) as a generic way to represent security-related information about the application object being accessed. More importantly, we proposed attribute retrieval function (AF) that serves attributes to DFs, as a new element to the updated, schema. By introducing ADME/AF schema, we made first step towards systematic support for application-specific factors in middleware access control mechanisms.

To illustrate the concepts of OSAs and AF on a concrete technology, we described their realization for CORBA Security. The described application of ADME/AF schema to CORBA domain has been adopted by the Object Management Group as part of SDMM specification [31] in November 2001, and it is currently on finalization track.

8 ACKNOWLEDGEMENTS

The concept of object security attributes has been prompted by a conversation with Ron Monzillo at one of OMG TC meetings in 2000. Their specific architecture and realization for CORBA Security have their origins from the work on SDMM specification that undergone numerous discussions. Some people who contributed most into these discussions and helped with their critiques of OSA architecture are (in alphabetical order) Ted Burghart, Fred Dushin, Don Flinn, Bret Hartman, Polar Humenn, Tadashi Kaji, Jishnu Mukerji, Mindy Rudell, and Kent Salmond.

Thanks to Bret Hartman for reviewing the paper in its early stage and providing helpful comments.

9 REFERENCES

[1] OSI, "Information Technology -- Open Systems Interconnection -- Security frameworks in open systems -- Part 3: Access control," ISO/IEC JTC1 10181-3, 1994.

[2] G. Karjoth, "The Authorization Service of Tivoli Policy Director," presented at Annual Computer Security Applications Conference (ACSAC), New Orleans, Louisiana, 2001, ACSAC 2001 CD.

[3] Netegrity, "SiteMinder Concepts Guide," Netegrity, Waltham, MA 2000.

[4] Entegrity, "Entegrity AssureAccess™ - Technical Overview," Entegrity Solutions September 2000, http://www.entegrity.com.

[5] Securant, "Unified Access Management: A Model For Integrated Web Security," Securant Technologies June 25 1999, http://www.cleartrust.com.

[6] P. A. Bonatti, E. Damiani, S. D. C. d. Vimercati, and P. Samarati, "A Component-based Architecture for Secure Data Publication," presented at Annual Computer Security Applications Conference (ACSAC), New Orleans, Louisiana, 2001, ACSAC 2001 CD.

[7] OMG, "CORBAservices: Common Object Services Specification, Security Service Specification v1.7," Object Management Group, document formal/01-03-08 2001, http://www.omg.org/cgi-bin/doc?formal/01-03-08.

[8] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers, "User Authentication And Authorization In The Java Platform," presented at Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, http://java.sun.com/security/jaas/doc/acsac.html.

[9] Sun, "Java Authentication and Authorization Service (JAAS),".: Sun Microsystems, 2001, http://java.sun.com/products/jaas/index-14.html.

[10] T. Ryutov and C. Neuman, "Access Control Framework for Distributed Applications (Work in Progress)," Internet Engineering Task Force, Internet Draft draft-ietf-cat-acc-cntrl-frmw-03, March 9 2000, http://www.ietf.org/internet-drafts/draft-ietf-cat-acc-cntrl-frmw-03.txt.

[11] T. Ryutov and C. Neuman, "Representation and Evaluation of Security Policies for Distributed System Services," presented at DARPA Information Servability Conference Exposition, Healton Head, South Carolina, 2000, http://www.isi.edu/gost/info/gaa_api.html.

[12] XACML-TC, "OASIS eXtensible Access Control Markup Language (XACML), Committee Draft," OASIS May 9 2002, http://www.oasis-open.org/committees/xacml/docs/.

[13] M. Kudo and S. Hada, "XML Document Security Based on Provisional Authorization," presented at ACM Conference on Computer and Communications Security, Athenes, Greece, 2000, D:\data\sources\security\XML security on provisional authorization -- Kudo and Hada.pdf.

[14] B. Hailpern and H. Ossher, "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Transactions on Software Engineering*, vol. 16, pp. 1247-1257, 1990.

[15] J. Barkley, "Implementing Role-based Access Control Using Object Technology," presented at The First ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1995, http://www.acm.org/pubs/citations/proceedings/commsec/270152/p293-barkley/.

[16] R. Filman and T. Linden, "SafeBots: a Paradigm for Software Security Controls," presented at New Security Paradigms Workshop, Lake Arrowhead, CA USA, 1996.

[17] T. Riechmann and F. J. Hauck, "Meta Objects for Access Control: A Formal Model for Role-based Principals," presented at New Security Paradigms Workshop, Charlottesville, VA USA, 1998, http://www.acm.org/pubs/articles/proceedings/commsec/283699/p17-riechmann/p17-riechmann.pdf.

[18] IETF, "RFC 1510, The Kerberos Network Authentication Service, V5,".: Internet Engineering Task Force, 1993.

[19] T. Parker and D. Pinkas, "SESAME V4 - Overview," SESAME December 1995, http://www.esat.kuleuven.ac.be/cosic/sesame/doc-ps/overview.ps.

[20] OMG, "Resource Access Decision Facility," Object Management Group OMG document number: formal/2001-04-01, August 2001, http://www.omg.org/cgi-bin/doc?formal/2001-04-01.

[21] K. Beznosov, Y. Deng, B. Blakley, C. Burt, and J. Barkley, "A Resource Access Decision Service for CORBA-based Distributed Systems," presented at Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, http://www.acsac.org/1999/abstracts/fri-b-0830-beznosov.html.

[22] HP, "HP Adds Value to DCE Security Framework with Praesidium Authorization Server," in *DCE application development trends Magazine*, 1996.

[23] R. Simon and M. E. Zurko, "Adage: An Architecture for Distributed Authorization," OSF Research Institute, Cambridge 1997, http://www.osf.org/www/adage/adage-arch-draft/adage-arch-draft.ps.

[24] T. Ryutov and C. Neuman, "Generic Authorization and Access control Application Program Interface: C-bindings," Internet Engineering Task Force, Internet Draft draft-ietf-cat-gaa-bind-03, March 9 2000, http://www.ietf.org/internet-drafts/draft-ietf-cat-acc-cntrl-frmw-03.txt.

[25] B. Hartman, D. J. Flinn, and K. Beznosov, *Enterprise Security With EJB and CORBA*. New York: John Wiley & Sons, Inc., 2001

[26] Encommerce, "getAccess Design and Administration Guide," Encommerce September 20 1999, http://www.encommerce.com.

[27] D. C. Schmidt, "Overview of CORBA,"., 2001, http://www.cs.wustl.edu/~schmidt/corba-overview.html.

[28] D. C. Schmidt and S. Vinoski, "Object Adapters: Concepts and Terminology," in *SIGS C++ Report*, vol. 9, 1997, http://www.cs.wustl.edu/~schmidt/PDF/C++-report-col11.pdf.

[29] OMG, "Specification of the Portable Object Adapter (POA)," Object Management Group formal/01-09-48, 2001.

[30] U. Lang, D. Gollmann, and R. Schreiner, "Verifiable Identifiers in Middleware Security," presented at Annual Computer Security Applications Conference (ACSAC), New Orleans, Louisiana, 2001, ACSAC 2001 CD.

[31] OMG, "Security Domain Membership Management Service, Final Submission," Object Management Group Document # orbos/2001-07-20, July 11 2001.

[32] K. Brown, *Programming Windows Security*, First ed. Upper Saddle River, NJ: Addison-Wesley, 2000