In Proceedings of the Annual Computer Security Applications Conference, Phoenix, Arizona, U.S.A., December 6-10, 1999

A Resource Access Decision Service for CORBA-based Distributed Systems

Konstantin Beznosov
Baptist Health Systems
of South Florida
6855 Red Road,
Miami, FL 33176
konstanb@bhssf.org

Yi Deng
Florida International University
University Park
Miami, FL 33199
deng@cs.fiu.edu

Bob Blakley
DASCOM
3004 Mission Street
Santa Cruz, CA 95060
blakley@dascom.com

Carol Burt
2AB
3178-C Highway 31 South, Pelham, AL 35124
cburt@2ab.com

John Barkley
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001
jbarkley@nist.gov

Abstract

Decoupling authorization logic from application logic allows applications with fine-grain access control requirements to be independent from a particular access control policy and from factors that are used in authorization decisions as well as access control models, no matter how dynamic those polices and factors are. It also enables elaborate and consistent access control policies across heterogeneous systems. We present design of a service for resource access authorization in distributed systems. The service enables to decouple authorization logic from application functionality. Although the described service is based on CORBA technology, the design approach can be successfully used in any distributed computing environment.

1. Introduction

Traditional access control mechanisms [1] provide limited capabilities for authorization decisions to be based on factors that are specific to the application domain. The complexity of access control policies in such application domains as healthcare requires exercising access control policies that are more sophisticated and of finer granularity than the general ones used in security services of such distributed environments as CORBA.[1] This complexity leads application designers to embed domain-specific authorization logic inside their applications. Some even document patterns of designing "application security" [2].

[1]Common Object Request Broker Architecture

CORBA environment, including the CORBA Security Service, provides a general-purpose infrastructure for developing distributed object systems in a broad range of specialized vertical domains. The CORBA Security service defines the interfaces to a collection of objects that provide a versatile set of services for enforcing a range of security policies using diverse security mechanisms. Some of these mechanisms require application systems to be aware of security. Such security models currently require application system designers to implement complex access control decisions based on content and context of interactions between client and target objects.

Security requirements in such a domain as healthcare mandate domain-specific factors (e.g. relationship between the user and the patient, emergency context) to be used in access control policies. At the same time, commonality of business domain tasks and security requirements across an enterprise computing infrastructure requires exercising fine-grained access control policies in a uniform and standard way.

This paper describes a CORBA-based authorization service, utilization of which allows fine-grain application-level access control in such a way that the functional design of application systems is separated from complexity and idiosyncrasies of particular enterprise access control policies. We show how decoupling of the authorization logic from application logic can be done if the described authorization service is used. In addition, our approach allows having a multi-policy authorization model, and it permits security administrators and application developers to maintain a clear separation of responsibilities.

The authorization service is by no means a replacement or substitution of standard CORBA Security service [3]. In fact, the concrete design proposed in this paper assumes existence and takes advantage of CORBA-compliant security infrastructure. More over, our solution is of general value and it is applicable to any distributed computing environment such as Sun RPC, DCOM, DCE or Java.

The design of the authorization service provides a way to have any level of access control granularity, allows integration with existing authorization models and systems, and supports such dynamic attributes as patient–caregiver relationships using existing authorization models. To achieve these benefits, our design requires application-level enforcement of authorization decisions and assumes agreement on semantics of resource names between the application developer and the owner.

This paper shows that decoupling of authorization logic from application can be done without complicated interactions between an application and the authorization service and without significant communication overhead. Factors specific to the application domain can be supported by authorization systems using the traditional access matrix as an underlying implementation. The body of the work described in this paper has been served as a foundation of the recently voted specification [4] of Resource Access Decision Facility from the Object Management Group. The initial design was prototyped and the current design has been implemented.

The rest of the paper is organized as follows: the next section provides an overview of CORBA security model and describes its access control model; Section 3 discusses the problems that we address in this paper; the service design is presented in Section 4; pros and cons of the design are discussed in Section 5; our approach is compared to related work in Section 6; the implementation status is reported in Section 7; we draw conclusions and discuss future work in Section 8.

2. Overview of CORBA access control model

CORBA environment, including the CORBA Security Service, provides a general-purpose infrastructure for developing and deploying distributed object-based systems in a broad range of specialized vertical domains. All entities in CORBA computing model are identified with interfaces defined in the OMG Interface Definition Language (IDL). A CORBA interface is a collection of three things: operations, attributes, and exceptions. An implementation of a CORBA interface is called a CORBA object. Hence, we use "CORBA object" or just "object" to mean "implementation of a CORBA interface", where it does not cause confusion. Object functionality is exposed to other CORBA-based applications only through the corresponding interfaces. Objects have object references by which they can be referenced. An object reference is a handle through which one requests operations on the object.

CORBA Security service (CS) 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 objects should be able to run securely on a secure ORB without any active involvement on the site of application objects. 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 because all messages between CORBA objects and clients are passed through the ORB.

CS uses the notion of principal. "A *principal* is a human user or system entity that is registered in and authentic to the system" [3]. In translation to the traditional security terminology, a principal is a subject. CS manages access control policies based on the security attributes of principals and attributes of objects as well as operations implemented by those objects. Objects that have common security requirements are grouped in security policy domains. Access control policies control what principals can invoke what operations on what objects in the domain the policies are defined on. Policies can be enforced either by the ORB or by the application. In the latter case, such an application is called a *security-aware application*. Domains allow application of access control policies to security-unaware objects without requiring changes to their implementations or interfaces.

As it can be seen in Figure 2, the client-side and target-side invocation access policy governs whether the client can invoke the requested operation on the target object on behalf of the current principal. This policy is enforced by the ORB in cooperation with the security service it uses for all (security-aware and unaware) applications. A client may invoke an operation on the target object as specified in the request only if this is allowed by the object invocation ac-

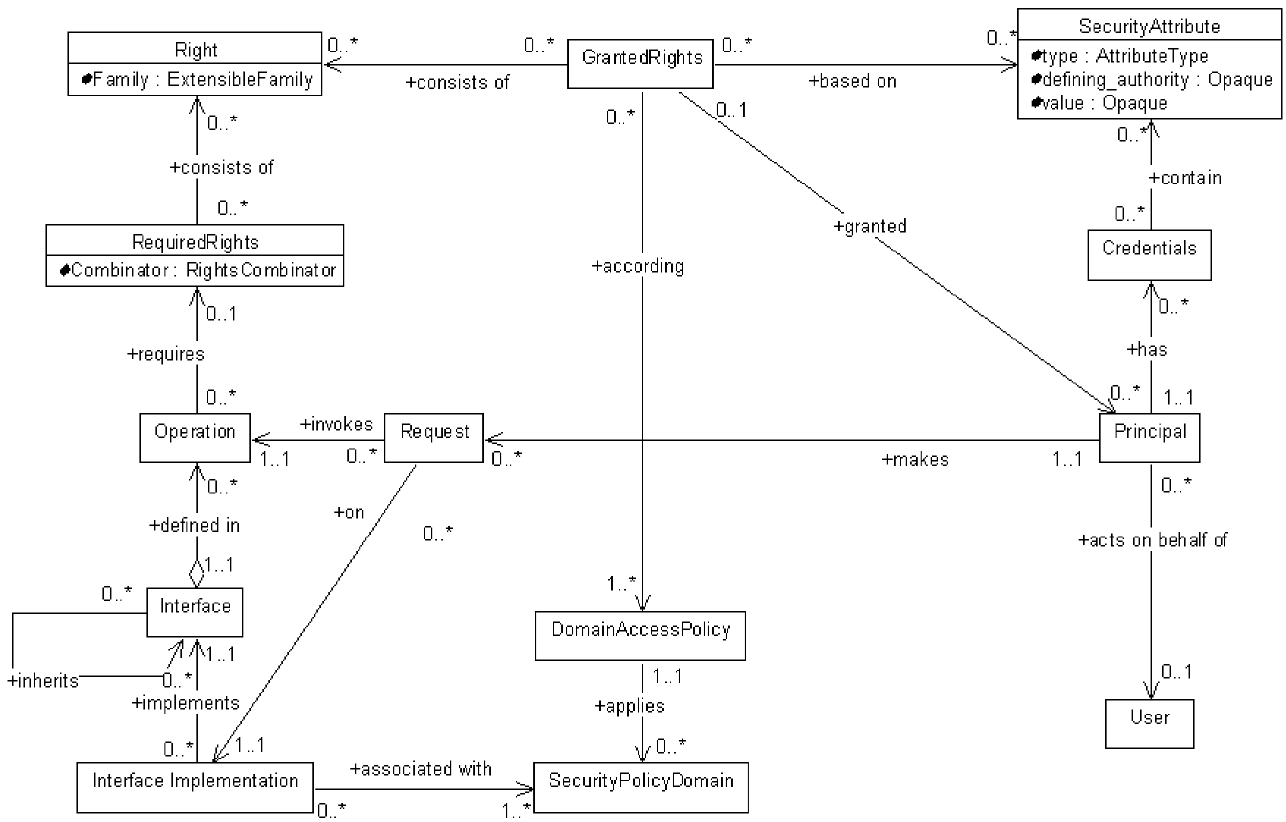

Figure 1. CORBA access control model

cess policy.

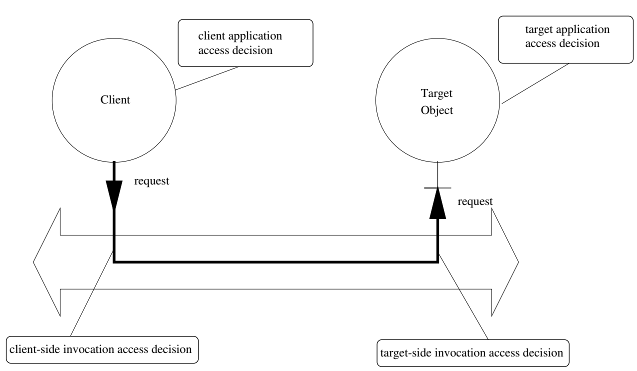

Figure 2. CORBA Security service access control model (from [3])

A user uses a *UserSponsor* to authenticate to the CS environment. A *UserSponsor* is an implementation artifact which handles user authentication process. After the user is successfully authenticated, a new principal with locality constrained *Credentials* object is created. The information in *Credentials* constitute the identity of the new principal which initiates requests on CORBA objects on behalf of the user. Principal authenticated security attributes are part of the information stored in *Credentials* object.

We provide an illustration of the following CS access control (AC) description in Figure 1. The concept of a user is absent from CS AC model. Instead a principal represents the user completely. The notion of a session is indistinguishable from the notion of a principal. Thus multiple principals can act on behalf of a single user. They all potentially have different sets of credentials and therefore exist in CS as completely independent entities. Among other data, principal credentials contain security attributes. Hereafter, we understand attribute to mean security attribute. From CS AC model point of view, a principal is nothing but an unordered collection of authenticated attributes. All attributes are typed. Attribute types are partitioned into two families: privilege attributes and identity attributes. The family of privilege attributes enumerates attribute types that identify principal privileges: access identifier, primary and secondary groups the principal is a member of, clearance, capabilities, etc. Identity attributes, if present, provide additional information about the principal: audit id, accounting id, and non-repudiation id, reflecting the fact that a principal might have various identities used for different purposes. Principal credentials may contain zero or more attributes of the same family or type.[2] Due to the extensibility of the schema for defining security attributes, an implementation of CS can support attribute types that are not defined by CORBA Security standard. Although the normative part of CS does not mandate the way attributes are managed, assignment of such attributes to users is meant to be done by user administrators.

All a principal does in the CORBA computational model is invoke operations on corresponding interface implementations. Such implementations are also called objects. Every object implements an interface. In order to make a request one needs to know two things: object reference, which uniquely identifies an object, and operation name. CORBA interfaces can inherit from other CORBA interfaces via interface inheritance. An operation name is unique for an interface[3] the object is implementing. Thus, any operation is uniquely identified by its name and by the name of the interface it is defined in. In this paper, we use notation $i_k m_n$ to refer to *n*-th operation on *k*-th interface.

There is a global, i.e. not dependent on a policy domain in which the object is located, set of rights (*RequiredRights*) for each operation. This set, together with a combinator (*all* or *any* rights), defines what rights a principal has to have in order to invoke the operation. It is assumed that required rights are defined and their semantics are precisely documented by application developers who know the best what each operation does. Depending on the access policy (*DomainAccessPolicy*) enforced in a particular AC policy domain,[4] a principal is granted different rights (*GrantedRights*) according to what *SecurityAttributes* it has. For the sake of brevity, we omit delegation state qualifier for granted rights. This does not change the correctness of the discussion, as we show below. Each *DomainAccessPolicy* defines what rights are granted for what security attributes. Security administrators are responsible for defining what rights are granted to what security attributes in what delegation state on domain per domain basis. Whenever a principal attempts an operation invocation, principal's effective rights are computed via operation *AccessPolicy::get_effective_rights*.[5] CS specification purposefully does not define how the operation combines rights granted through different privilege attribute entries. The specifiers let CS implementers to define the operation internal behavior ([3, p. 122]). A simplest implementation of *get_effective_rights* could be when the set of rights granted to a principal is a union of rights granted to every security attribute the principal has.

[2]This rule applies to all attribute types including access id, although it is hard to foresee a useful implementation of CS where a principal would have multiple or no access identities.

[3]Interface inheritance in CORBA does not allow to inherit from interfaces with operations of the same type. This rule resolves the problem of operation name overloading.

[4]In CORBA security model, a security policy domain is just a collection of objects.

[5]Regular caching techniques can be used by an implementation to avoid repetitive computations.

3. Problem description

This section shows why there is a need in security-aware implementations of CORBA objects to enforce their own access control policies, as well as problems with embedding such control into application systems.

3.1. Why application-level access control

There are two main reasons for application-level access control, namely the necessity in fine-grain access control and the need for authorization decisions based on factors that can be "known" only to the application.

Fine-grain access control is necessary because sometimes the sensitivity of the information accessed via the same *operations*[6] of a CORBA service interface differs. In healthcare for instance, different parts of the patient medical record have different levels of sensitivity. Obvious examples are patient name and HIV-related test data.

Another crucial reason for application-level access control is the need in using application domain-specific factors in authorization decisions. Analyses made by one of the authors and discussed elsewhere [6], [7] reveals the necessity of sophisticated access control policies in healthcare systems. They are due to the various legal and liability requirements imposed by state and federal legislation [8]. Ideally, authorization decisions in the healthcare domain should be based on the following factors [9]: subject affiliation, subject role, subject location, access time, and relationship between the subject and the patient whose records are to be accessed.

Relationship is a good example of an authorization decision factor, which is specific to the healthcare vertical domain. Its value ideally should be derived from the information scattered across various clinical, billing, and patient registration systems. Some types of relationships that need to be managed in the healthcare context are: patient's primary care provider; admitting, attending, referring, or consulting physician of a particular patient; part of the patient care team; healthcare staff explicitly assigned to take care of the patient; patient's immediate family; patient's legal counsel or guard; personal pastoral care provider. The relationship factor is very dynamic and ideally it should be computed dynamically every time a decision is made. We expect that other vertical domains have similar requirements

[6]Operation is a synonym to method in OO terminology. We use it according to the object management model [5] from the OMG.

in access control policies regulated by domain-specific factors that cannot be modeled using groups, roles, or identities.

3.2. Problems with authorization logic embedded in application systems

Since the application programmer understands the application functionality most intimately, building authorization logic into the application allows the application to control access at an arbitrary granularity level and to use authorization rules of an almost unlimited complexity. However, authorization logic coupled with application logic produces serious consequences. Embedding authorization logic into application systems causes problems that can be qualified as software engineering and information enterprise security administration. This paper discusses problems related to operation and administration of enterprise security.

With authorization logic embedded into application systems, enterprise security administrators end up having to configure such access logic on an application-by-application basis, which brings tremendous administrative overhead and highly increases chances of human error. Because each application system has its own access control model, which is administrated via proprietary interfaces, multiple inconsistent security authorization models co-exist in the same information enterprise. It is difficult to ensure consistency of authorization policies across the enterprise. Most of the time, security administrators end up having no guarantee, whatsoever, that access rules and, especially, changes to them are consistent across all application systems as well as with required company policies. In addition, an environment with mixed authorization and application logic merges an administrator's responsibilities with an application developer's responsibilities and vice versa.

The approach presented in the next section permits security administrators and application developers to maintain a clear separation of responsibilities, as well as to avoid most of the software engineering shortcomings of embedding authorization logic in the application.

4. Resource access decision service

In this section, first we describe the scope of the authorization service and the interactions between the service and application systems. Then, we describe the design of the authorization service.

As it was shown in Section 2, the granularity of CORBA access control mechanisms is at the level of operations on CORBA objects. The authorization service is to make authorization decisions for access to those information and computational resources by CORBA services that are not first class CORBA objects and their operations, as shown in Figure 3. Thus, the service complements CORBA security access model. It relies on and uses CORBA security environment.

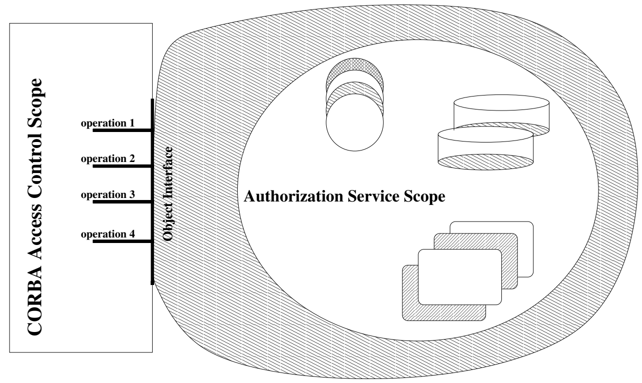

Figure 3. **Scope of the authorization service**

4.1. Interaction Between Application Service and Authorization Service

The main objective of RAD is to decouple application-level authorization logic from application logic. Authorization logic is encapsulated into an authorization service external to the application, which is traditionally part of an application program. A simplified schema of interactions among application client, application service and an instance of authorization service is depicted in Figure 4. To perform an application-level access control, an application requires an authorization decision from such a service and enforces that decision. Simple interfaces between the application and the authorization service are used, where an application programmer only needs to make a single invocation on the authorization service in order to obtain a decision.

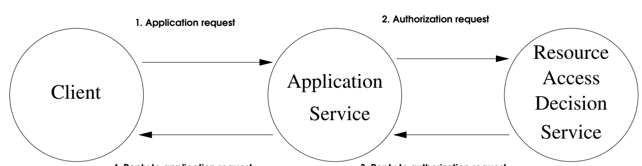

Figure 4. **Interactions between client, application system, and authorization service.**

The sequence of the interaction, illustrated by Figure 4, is as follows:

1. An application client invokes an operation on the application service (application, for short).

2. While processing the invocation, the application requires an authorization decision from the RAD.

3. The RAD makes an authorization decision, which is returned to the application.

4. The application, after receiving an authorization decision, enforces it. If access was granted by the RAD, the application returns expected results of the invocation. Otherwise, it either returns partial results or raises an exception.

An application obtains an authorization decision only from one instance of RAD. It is the contract between the application and its enterprise environment to request an authorization decision and to enforce it. Before we proceed with greater details on the design of an authorization service, we will describe syntaxes and semantics of a request for authorization decision.

From RAD perspective, any application requesting an authorization decision is an RAD client. From now on, we will use the term "RAD client" to refer to any entity of the distributed system that requested an authorization decision from an RAD.

A nominal amount of data is passed between the application and the authorization service in order to make authorization decisions. When making a request for an authorization decision, an RAD client passes the following three parameters: a sequence of name-value pairs representing a name of the resource to be accessed on behalf of the client; name of access operation (e.g. "create", "read", "write", "use", "delete"); authenticated security attributes of the subject on behalf of which the client is requesting access to the named resource.

Security attributes here are regular attributes of the current user session. The interesting parameters passed by RAD client are the first two: resource name and access type. They are described below.

We introduce an abstraction called "protected resource name" or just "resource name." Resource name is used to abstract application-dependent semantics and syntaxes of entities under application-level access control. A resource name can be associated with any valuable asset of an application owner, which is accessed by a client on behalf of a subject using it, and access to which is to be controlled according to the owner's interests. For example, electronic patient medical and billing records in a hospital are usually its valuable assets. The hospital administration is interested in controlling access to the records due to various legal, financial and other reasons. Therefore, the hospital administration considers such records as protected resources. Moreover, different information in those records count as different resources. Examples of different resources can be records from different visits or episodes for one patient. At the same time, a resource name can be associated with less tangible assets, such as computer system resources, including CPU time, file descriptors, sockets, etc. The RAD does not attempt to interpret semantics of the resource name. We will show in the discussion of the RAD design that it uses the resource name only to obtain additional security attributes and to look up a set of policies that govern access to the resource associated by an application system with the resource name.

Access operation abstracts semantics of access to resources associated with resource names. An application may manipulate with patient records on behalf of different care-givers, or may provide different hierarchies of menus to different technicians of the hospital lab. In either case, it is up to the application system developers and the enterprise security administrators to agree on semantics of the operation name used for each access. The RAD does not interpret semantics of access operation as it is shown in the description of the RAD design.

Before an application requests an instance of RAD for authorization decision, it is supposed to identify what the resource name and the access operation name are associated with servicing the client request. There is not any particular algorithm defined for performing such an association. For every application, or at least for every application domain, the way of associating protected entities with abstract resource names can be different.

4.2. Design of the service

RAD service is composed of the following objects[7]: *AccessDecisionObject* (ADO) receives requests on authorization decisions from RAD clients. Zero or more *PolicyEvaluators* provide evaluation decisions for those policies that govern access to the given resource. If a policy evaluator does not have any policy associated with the given resource name, the evaluator returns a result meaning "don't know," therefore delegating the decision combinator to apply its combination policy while combining results from potentially several evaluators, depending on the combinator configuration. *PolicyEvaluatorLocator* keeps track of and provides references to potentially several policy evaluators. *DynamicAttributeService* provides dynamic attributes of the principal in the context of the intended access operation on the given resource associated with the provided resource name. *DecisionCombinator* combines results of the evaluations made by policy evaluators into a final decision by resolving evaluation conflicts and applying combination policies.

Figure 5 shows the interaction among the parts of the authorization service. Once the authorization service received

[7]Since in OMA a service entity can implement multiple interfaces, and objects are nothing else but implementations of interfaces, we refer here to an object to signify a particular interface implementation. An implementation of the authorization service described here can implement any number of the specific interfaces in one entity of the CORBA environment.

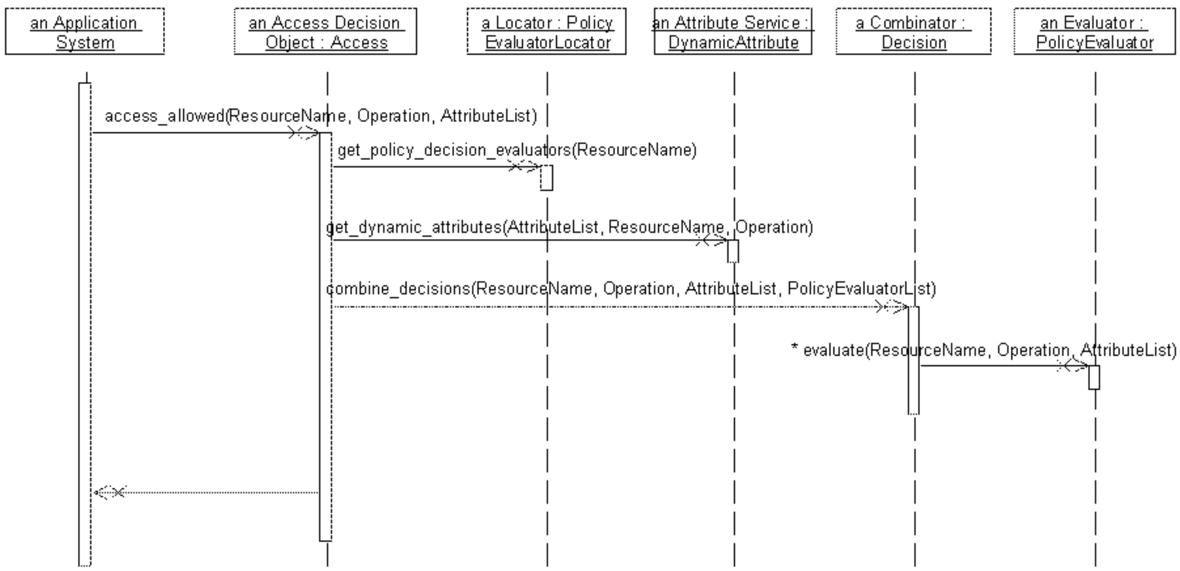

Figure 5. Interaction diagram of the authorization service components

a request via the ADO interface: *ADO* obtains object references to those *PolicyEvaluator*s that are associated with the resource name in question and an object reference for the *DecisionCombinator* which will combine the decisions. *ADO* obtains dynamic attributes of the principal in the context of the resource name and the intended access operation on it. *ADO* delegates an instance of *DecisionCombinator* for polling *PolicyEvaluator*s (selected in step 1). A *DecisionCombinator* obtains decisions from *PolicyEvaluator*s and combines them according to the combination policy. The decision is returned to *ADO*. The *ADO* returns the decision to the application.

4.3. Dynamic security attributes

One of the significant points of the design is handling the factors specific to the application domain in the manner neutral to their semantics. All such factors are handled as dynamic attributes. We introduce qualifier "dynamic" to distinguish them from regular privilege attributes of the subject, which we call "static attributes" here. They are obtained from the enterprise environment via specialized dynamic attribute services. An authorization service does not interact with such services directly. It delegates the generic dynamic attribute service to collect all dynamic attributes from specialized services. The semantic of a particular application domain (patient – care-giver relationship) can be expressed in the form of dynamic attributes. This allows utilization of already existing authorization mechanisms such as the traditional access matrix [10].

Dynamic attributes are those attributes that express properties of a principal but are not administrated by security administrators. A user usually has dynamic attributes due to the various activities the user performs in the enterprise work-flow. Dynamic attributes are so called because their values usually change more frequently than traditional user privilege attributes. Traditional "static" security attributes are used for describing relatively fixed properties of users and/or resources. The values of static attributes are typically set by security administrators and are obtained by an application in an environment specific manner, e.g., from a principal's credentials in case of CORBA environment. While the use of a dynamic attribute in an access decision is determined by a security administrator, the values of dynamic attributes are usually set as part of normal processing, i.e., dynamic attribute values are usually part of information content not separately maintained security meta-data. Consequently, dynamic attribute values must be obtained at the time an access decision is required. This is in contrast to traditional "static" privilege attributes whose values are usually obtained when a session is established. The values of dynamic attributes may change during a session as a result of normal work-flow processing.

Consider the following example of a dynamic security attribute. Physician John Smith attends patient B. The physician has an attribute specifying such a relationship when principal with access_id=*johnsmith* (speaking for John Smith) is accessing resources associated with medical records of patient B. This relationship attribute is an example of a dynamic attribute in our model. It has the value "attending physician" returned by a generic DAS only when John Smith accesses B's records. The generic DAS obtains the value of this relationship attribute by consulting a specialized DAS, which has capabilities to compute the value of relationship attribute. For instance, by looking at the corresponding fields of B's patient record which contains a list of B's attending physicians. When John Smith is accessing resources not associated with any patient, this dynamic attribute of type *relationship* is not returned by the corresponding specialized DAS and consequently it is not returned by generic DAS.

4.4. Policy Evaluators

Another significant design element is encapsulation of authorization policies and their evaluators into separate entities in the computational environment. Policy evaluators can be considered either as distinct authorities each representing a different set of authorization policies, or they can be considered as policy evaluation machines each supporting a particular policy language. Such design insulates representation and interpretation of policies from the authorization service. It also allows adding and removing policy evaluators dynamically. By encapsulating the evaluation of those policies in *PolicyEvaluator* objects, the design supports implementation of arbitrary authorization policies.

4.5. Separation of concerns

Separation of concerns among various stake-holders involved in the authorization process enables control of different factors in the authorization process by appropriate parties. Application developers decide what functions of their application map into what access operations. User administrators control what users (or roles) are assigned what static security attributes. Implementors of the authorization services and other third party vendors control quality, performance and other properties of the authorization service implementation. Work-flow administrators indirectly control what dynamic attributes are assigned to what users in the context of what resources. Security administrators administrate what access control policies govern what access to what named resources.

5. Discussion

Our solution has the following advantages:

Simplicity. Simple interfaces between the application and the authorization service are used. An application programmer is required to make a single invocation on the authorization service in order to obtain a decision. All required information is represented by such simple structures as resource names, operation names, and principal security attributes. A nominal amount of data is passed between the application and the Authorization Service in order to make authorization decisions.

The programming complexity of making authorization decisions for an individual policy is encapsulated in *PolicyEvaluatorLocator, DynamicAttributeService,* and *PolicyEvaluator* objects. Thus, simple policies allow overall simplicity of the model. The complexity increases only by introducing complex types of authorization policies and sophisticated *specialized DynamicAttributeServices*. *PolicyEvaluatorLocator* can be as simple as an implementation of relational table indexed by resource name.

Generality. Due to the design, the authorization service can be utilized in various application domains. It introduces the notion of resource name, which in its turn allows arbitrary granularity of protected resources. The application system decides, depending on the application domain, how small the unit of access control is. The resource name, principal security attributes as well as request dynamic attributes, and the intended operation name should communicate any semantic information that can be used for applying reasonable[8] authorization policies. The design supports arbitrary authorization policies by encapsulating the evaluation of those policies in *PolicyEvaluator* objects.

Flexibility. Due to the use of CORBA infrastructure with object implementation location transparency and its services such as Naming and Trader, the proposed design enables implementations adaptable to changes in authorization policies and their types as well as in the workflow of the user organization via replacement of *PolicyEvaluators* and *specialized DynamicAttributeServices*. New *PolicyEvaluators* can be registered with the *PolicyEvaluatorLocator* and new *specialized DynamicAttributeServices* can be registered with the *DynamicAttributeService* object or obtained via CORBA Naming or Trader services. The semantic of a particular application domain (patient–caregiver relationship) can be expressed in the form of dynamic attributes. This allows utilization of already existing authorization mechanisms such as the traditional access matrix. Separation of concerns among various stake-holders involved in the authorization process enables control of different factors in the authorization process by appropriate parties.

There are three main issues with the proposed approach. It is not clear whether it is possible to abstract all protected resources into resource names. The proposed solution requires such abstraction. Matching in dynamic attribute semantics between policy evaluators and specialized dynamic attribute services has to be maintained. One of the ways to reduce performance penalties of obtaining a decision from an authorization service is to co-locate an application system and an authorization service. Simple co-location increases the number of authorization service instances to ad-

[8] We do not define here what policies fall in the scope of reasonable ones. We think it is the subject of separate research, which we describe in Section 8.

ministrate. On the other hand, an optimum administration solution would be such that it requires to administrate only one instance of administration interfaces. Current design of the authorization service does not provide ways to have a single set of administration objects and multiple instances of authorization services.

There are also implementation issues that have to be addressed in order to develop an efficient and scalable implementation. One of them is proper parallelization in order to avoid bottlenecks. The back-end data needed by PolicyEvaluators and DASs could become a bottleneck in accessing authorization service, when multiple ADO clients consult instances of ADOs. This could decrease scalability of the system. Regular caching and replication techniques should be sufficient for maintaining system scalability.

6. Related work

The ideas of discretionary access control (DAC) model proposed by Lampson in [10] has led to the concept of a reference monitor outlined by Anderson in [11]. When an application enforces its own access control policies, a reference monitor is embedded in the application. Our authorization framework allows externalization of a reference monitor from an application without losing the capability for an application to define its own space of protected resources and its semantics.

Abadi et al. [12] and Lampson et al [13] developed a unified theory of authentication and access control in distributed systems. Practical implementations reflecting some results of the theory have been implemented in security architectures of such distributed environments as DCE [14], DCOM, and CORBA [3]. Our work suggests an authorization framework for implementing multiple fine-grain and workflow-dependent access control policies in application systems developed for such environments. Even though we present a concrete solution that uses CORBA security infrastructure, the underlying schema should be implementable for DCE and DCOM, because the only requirement for the underlying security infrastructure is the capability of an application to query the infrastructure for the principal security attributes of the client.

Multi-policy authorization paradigms and frameworks have been proposed by a number of research projects ([15], [16], [17], [18]). They use an object method in Argos [17] or a database table record in [18] as the finest level of access control decisions. In our approach, the authorization decision is obtained after the method on the object is invoked. Hence, an application can exercise access control of any granularity level by associating a resource name with protected elements of any size and semantics. One reference monitor (supporting a particular policy) per request is used in Argos to evaluate requested access. Due to introduction of multiple evaluators and a combinator, we provide ways for more than one policy (of different types), as in Bertino et al. [18], to govern authorization decisions for the same request. Bertino and Jajodia in [18] define an explicit authorization model with conflict resolution and overriding rules. Such rules have to be implemented by a particular instance of decision combinator in our framework. This is left as future work for our framework.

The proposed concept of dynamic attribute service gives enough flexibility in using enterprise-specific factors to support all implicit access rights that Argos does as well as PICASSO's [19] patient-specific roles of the principal and other types of access rights. Our approach allows Argos and PICASSO policy engines to be used as one of the policy evaluators in the authorization service described here. This would be similar, although not exactly the same, to what Johnscher and Ditrich suggest in [17] when they write that "Argos can be used as an access control service for any application that is connected to the corresponding object request broker."

7. Implementation status

A prototype of the first version of the authorization service design has been implemented by 2AB, Inc. and is available at http://www.omg.org/docs/corbamed/99-01-19.zip. It includes the implementation of the authorization service with interfaces as defined in [20], a policy administration system necessary to allow resources and policies to be defined, and a client program to test sample policies. A functioning prototype of the design outlined in this paper and specified in details in [4] has been implemented at the Center for Advanced Distributed Systems Engineering (CADSE)[9] of Florida International University.

8. Conclusions

In this paper we presented an approach in decoupling authorization logic from application logic for those CORBA-based application systems, which resort to application-level access control in order to achieve fine granularity of protection or to use factors specific to the application domain in authorization decisions, or both. We described the design of an authorization service that allows any level of access control granularity, applying authorization policies of different types and from different authorities, as well as providing application domain-specific factors for evaluating such policies.

The following two results are the main contributions of the paper: (1) decoupling access control from applications can be done without complicated interfaces and without

[9] http://cadse.cs.fiu.edu

sending much information between an application and the authorization service; and (2) dynamic attributes, such as the patient–caregiver relationship, can be supported using a traditional access matrix as an underlying implementation.

The body of the work described in this paper has been served as a foundation of the recently voted specification [4] of Resource Access Decision Facility from the Object Management Group.

We plan to show what types of policies can be supported by the proposed design effectively, to develop a more precise specification of the authorization service, and to obtain experimental data on performance and scalability of the described solution.

References

[1] Ravi Sandhu and Pierangela Samarati. Access control: Principles and practice. *IEEE Communications*, 32(9), September 1994.

[2] Joseph Yoder and Jeff Barcalow. Application security. In *Proceedings of The 4th Pattern Languages of Programming Conference*, 1997.

[3] Object Managment Group. *CORBAservices: Common Object Services*, July 1998. OMG document number: formal/98-07-05.

[4] Object Management Group. *Resource Access Decision Facility*, May 1999. OMG document number: corbamed/99-05-04.

[5] Richard Mark Soley and Christopher M. Stone. *Object Management Architecture Guide*. John Wiley & Sons, 3 edition, June 1995.

[6] Wayne Wilson and Konstantin Beznosov. *CORBAmed Security White Paper*. Object Management Group, November 1997. OMG document number: corbamed/97-11-03.

[7] Konstantin Beznosov. Issues in the security architecture of the computerized patient record enterpirse. In *Proceedings of Second Workshop on Distributed Object Computing Security*, Baltimore, Maryland, USA, May 1998. The Object Management Group and the United States National Security Agency.

[8] Konstantin Beznosov. Taxonomy of CPR enterprise security concerns at Baptist Health Systems of South Florida. http://www.bhssf.org/IT/Projects/cpr/security/progress-reports/categorize-requirements.html, December 1997.

[9] Konstanantin Beznosov. Requirements for access control: US healthcare domain. In *Proceedings of the Third ACM Workshop on Role-Based Access Control*, page 43. Fairfax, Virginia, USA, October 1998.

[10] Butler Lampson. Protection. In *In 5th Princeton Symposium on Information Science and Systems*, pages 437–443, 1971.

[11] James Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Vols. I and II, Air Force Electronic Systems Division, 1972. NTIS document number AD758206.

[12] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. Technical Report 70, DEC, March 1991.

[13] Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practics. Technical Report 83, DEC, February 1992.

[14] Open Software Foundation, 11 Cambridge Center Cambridge, MA 02142. *OSF DCE Application Development Guide: Core Components*, 1.2.1 edition, 1996.

[15] Dobson J. and McDermid J. A framework for expressing models of security policy. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 229–239, May 1989.

[16] Hosmer H. Multipolicy paradigm. In *Proceedings of the New Security Paradigm Workshop*, Little Compton, RI, 1992.

[17] Dirk Jonscher and Klause R. Dittrich. Argos – a configurable access control system for interoperable environments. In *Proceedings of the IFIP WG11.3 Ninth Annual Working Conference on Database Security*, pages 39–66, Rensselaerville, NY, 1995.

[18] Bertino E., Jajodia S., and Samarati P. Supporting multiple access control policies in database systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1996. IEEE Computer Society Press.

[19] Dixie B. Baker, Robert M. Barnhart, and Teresa T. Buss. PCASSO: Applying and extending state-of-the-art security in the healthcare domain. In *Annual Computer Security Applicatications Conference*, 1997.

[20] Object Management Group. *Healthcare Resource Access Control (Initial Submission)*, October 1998. OMG document number: corbamed/98-10-02.