OMG CORBAmed DTF

Healthcare Resource Access Control (HRAC)

Initial Submission

2AB, INC.
Baptist Health Systems of South Florida
CareFlow|Net, Inc.
IBM

OMG TC Document corbamed/98-10-02
18 October 1998

The submitting companies listed above have all contributed to this "initial" submission. These companies recognize that this initial submission is the joint intellectual property of all the submitters, and may be used by any of them in the future, regardless of whether they ultimately participate in a final joint submission.

The companies listed above hereby grant a royalty-free license to the Object Management Group, Inc. (OMG) for worldwide distribution of this document or any derivative works thereof, so long as the OMG reproduces the copyright notices and the below paragraphs on all distributed copies.

The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. The companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

History

Initial Submission Draft 1	18 October 1998
The Editor of this document is Carol Burt of 2AB. The initial submission to the OMG Healthcare Resource Access Control Facility (HRAC) is the result of collaboration between the four submitting companies listed on the cover and the list of supporting companies named in the initial submission.	

Table of Contents

1. Preface

This submission is a response to CORBAmed RFP, Healthcare Resource Access Control (HRAC), Object Management Group (OMG) document number corbamed/98-02-16.

1.1 Submission Contact Points

Carol Burt 2AB 3178-C Highway 31 South Pelham, AL 35124 205 621 7455 cburt@2ab.com	Konstantin Beznosov Baptist Health Systems of South Florida 6855 Red Road Coral Gables, FL 33143 305 596 1960 beznosov@baptisthealth.net
V. “Juggy” Jagannathan CareFlow|Net, Inc. 235 High Street, Suite 225 Morgantown, WV 26505 304 293 7535 juggy@careflow.com	Bob Blakley IBM 11400 Burnet Road, Mail Stop 9134 Austin, TX 78756 512 838 8133 blakley@us.ibm.com

1.2 Supporting Organizations

The following organizations have been involved in the process of developing, prototyping and/or reviewing this submission. The submitters of this response thank them for participating and giving their valuable input. A special thank you goes out to those organizations. The editor would like to extend a special thanks to John Barkley of NIST for providing text and a detailed review of each version of this submission.

- Concept Five
- Inprise
- Los Alamos National Laboratory
- National Institute of Standards (NIST)
- National Security Agency (NSA)
- Philips Medical Systems

1.3 Conventions

```
IDL appears using this font and in a border.
```

1.4 Terminology[A]

Access Decision Object (ADO) – the initial request for an access decision is passed to the Access Decision Object. From the perspective of a client requesting an access decision of HRAC, this is the only interface that they are required to use.

Access Policy – the policy or rules that govern access to a secured resource.

AttributeList – a list of Priviledge attributes that are used to determine whether access should be allowed. An AttributeList may contain both static and dynamic attributes.

Client - Any system or application that accesses or requests service from the HRAC

Component - A cohesive set of software services

Dynamic attribute – a priviledge attribute that can only be determined at the time an access decision is requested. Dynamic attributes are often based on the relationship between a principal and the secured resource (such as attending physician) and cannot be statically configured. This submission allows dynamic attributes to be discovered dynamically and used as part of the access decision criteria.

Naming Authority - Any organization that assigns names determines the scope of uniqueness of the names and takes the responsibility for making sure the names are unique within its name space. In the same way that ID values are meaningful only within the context of their ID Domains, names are unique only within the context of their *naming authority*.

Principal: A principal is a human user or system entity that is registered in and authentic to the system

Priviledge Attributes – A principal may have priviledge attributes that can be used to decide what it can access. A variety of priviledge attributes may be available depending on access policy.

Secured Resource – a "secured resource" is any valuable asset of an application owner, which is accessed by an application on behalf of a principal using it, and access to which is to be controlled according to the owner's interests.

Static Attribute – a Priviledge attribute that is (typically) statically configured by an administrator. Examples would be access_id:john_doe or role:physician.

System - An application or set of applications that interact with each other, interact with the HRAC or implement HRAC. *System* in this context is synonymous with *application*. Examples of systems might include a hospital or clinical information system, an ancillary system such as a lab or radiology system, or a financial/administrative system such as an ADT.

1.5 Proof of Concept

The initial submission is based on experience gained in implementation of proprietary access control systems by 2AB, Concept Five, and IBM and with requirements input from end user organizations such as Baptist Health Systems of South Florida, and Healthcare vendors such as Phillips Medical Systems and CareFlow|Net. The interfaces in the initial submission will be prototyped by at least one submitting organization prior to the final submission.

[A] Where security terminology is presented here that is identical to terminology in CORBAsecurity, it is intended to be consistent. The CORBASec document is OMG document ptc/98-02-01

1.6 Changes to Adopted OMG Specifications

No changes to the existing OMG specifications are needed by this specification.

1.7 Response to RFP Requirements

1.7.1 MANDATORY RFP REQUIREMENTS

Use of the CORBA Security service credentials as the source for identifying caregivers' priviledges

The AccessDecision Interface takes a Security::AtrributeList as the source for identifying caregivers' priviledges. This attribute list is directly accessible from the Security::Credentials. The Credentials is not passed because the AccessDecision object is not locality constrained and to pass the object reference of a locality constrained object would violate CORBA Security. It would, however, be possible for a locality constrained SecurityLevel2::AccessDecision object to extract the Security::AttributeList from the Security::Credentials and use the HRAC AccessDecision object as part of an access control implementation.

Ability to define secured resource categories.

The ResourceName is a sequence of string where the first string in the sequence is required to be a NamingAuthority::QualifiedNameStr. This allows an implementation to provide groupings of secured resources and for a client to determine from the resource name how those groupings are arranged into hierarchies. The use of the NamingAuthority module allows these groupings to be unique.

An interface for defining access control rules for secured resources based on credentials.

The DecisionRuleAdmin interface provides this capability. Rules are based on the rights associated with the SecAttributes that are part of the credentials. Rules are expressed as sequences of rights which may be required to exist in combinations. The effective rights of the SecAttribute may be a one-to-one mapping from the attribute or may be obtained from a SecurityAdmin::AccessPolicy get_effective_rights() method of CORBASec implementation.

A set of Healthcare specific secured resources.

The initial submission does not currently provide this set. The final submission will contain a set of standard set of ResourceName constants.

An interface to an access control decision facility that may be used to request access control decisions.

The AccessDecision interface provides this capability

1.7.2 OPTIONAL RFP REQUIREMENTS

Provide the ability for secured resources to be grouped for the purpose of defining access control rules

This submission supports the concept of grouping in the ResourceName. It does not mandate the way an implementation interprets these grouping or the relationship between the **decision rules** for a resource that represents a group and the decision rules of resources within the group.

An interface for defining access control rules based on attributes of the Principle (in addition)

The specification defines how decision rules can be constructed which achieve this within the required rights model of CORBASec. A one-to-one mapping from a SecAttribute to a Right would be defined. ISSUE: The initial submission does not specify how an implementation would make this mapping, but it is under consideration whether this would be a good idea for a revised submission to standardize this mapping for interoperability purposes.

An interface that extends the definition of access control rules to include context sensitive access control based on a) the day and time when the resource is accessed, b) the location of an invoking principal, c) the values of request parameters.

The decision rules interface will allow time based rules and the DecisionRulesAccess is based on effective rules so that those not in effect may be masked by the implementation. The specification also allows dynamic rights that are evaluated by a DynamicEvaluator object. The DynamicEvalutator may be provided by the implementation, the application, and/or the Enterprise in which the HRAC is deployed. Thus the submitters feel they have provided the architectural foundation for context sensitive access control external from the application.

An interface that extends the definition of the access control rules to include notion of the relationship between a patient and a caregiver

The DynamicEvaluator interface was designed specifically as a generic way to support rrelationship based (and other dynamic rights based) components of decision rules. When a right is dynamic (such as the relationship between a patient and caregiver), it is expressed as a dynamic Right and is used in defining the decision rules. A DynamicEvaluator is provided for the Right will be consulted at the time of access decision by the AccessDecision object. See discussion of DecisionRulesAdmin set_dynrights_support() and DynamicEvaluator has_right().

A reference object model for the healthcare domain that provides a sufficient foundation for access decision logic.

The object model is not formally expressed in UML in this initial submission as it would require work to also formally express the CORBASec model upon which this submission depends. As in CORBASec, the object model is currently expressed in diagrams and text.

An interface that permits management of policy, which controls how multiple access control policy decisions governing access to the same resource are reconciled.

The submitters felt that such expression of policy was outside the scope of the submission.

1.7.3 DISCUSSION POINTS REQUESTED

How new CORBAMed specifications will employ the submitted specification . (Konstantin's words)

Due to the generality of the submitted response, the specified service can be used in various ways. The usage patterns will highly depend on a particular enterprise, its work-flow and access control policies. Usage of the service even in systems compliant with CORBAmed specifications is expected to vary from company to company. On the other hand, the submission team believes that semantics of the interactions between an HRAC service and its clients should be defined completely and precisely. It is the intend of the submission team to provide in further revisions of this response a complete and precise definition of the semantics. Besides defining semantics of interactions between HRAC service and its clients, the response provides sample scenarios and use cases. Sections 2.8 (General Usage Discussion) and 2.9 (Healthcare Specific Usage Scenarios) of the response provide such scenario's and use cases along with discussion about how the specified service is intended to be used in general cases and specifically in healthcare applications compliant with the OMG standards.

How existing CORBAMed specifications are to be modified.

The submitters do not believe any modification is necessary for existing CORBAMed specifications to use the services of an HRAC, however it might be useful for some standard ResourceNames to be defined within the CORBAMed community for common resources within standard services. Such definition would be a compatible extension of existing specifications.

Scalability and Performance of the proposal

By its nature, disposing requests for authorization decision, each time a separate entity, associated with a secure resource, is accessed, is an expensive and inefficient action, comparative to the existing CORBA access control mechanisms. The most efficient and appropriate way to implement access control for application systems is to use CORBA security service, which is intended to provide efficient and scalable access control enforcement. The intended role of HRAC service is not to replace access control mechanism available in CORBA security but to use it in addition to CORBA security. HRAC service is intended to be used in those cases when granularity and/or expressiveness of CORBA security access control model is insufficient. Thus, the submission team believes that use of HRAC service is a necessary "evil" in terms of overall system performance. Performance decrease can be eliminated but it cannot be lower than for use of CORBA security service only as in any security-aware application.

Taking the above discussion into account, the submit ion team believes that the there is anything in the proposed design model of HRAC service that would preclude implementation and deployment of scalable and having high performance HRAC services. Scalability and performance of systems implementing the proposed specification and usage of it by other CORBA-compliant systems are highly dependent on the following factors:

- Internal design and implementation of the HRAC-compliant service.
- Co-location of HRAC services and HRAC clients.
- Distribution of load over multiple instances of the HRAC services.
- Organization of the resource space.
- Complexity of access control policies.

All these factors might substantially affect overall system scalability and performance. The submitted design model does not preclude scalable and efficient execution of the all items in the list above. The proposed design implies a restriction only on the resource space: it has to consist of either independent atomic resources and/or a forest of resource trees. The submission team believes such an organization is generic enough and allows efficient and scalable implementations of resource space. There are more detailed description and discussion on the resource space organization in section [section number] "Section name" on page [page number].

1.7.4 Mechanisms provided for extensibility

To be written

2. Overview of Response

2.1 Introduction

This document is a response to the Healthcare Resource Access Control RFP (corbamed/98-02-23). The response describes a specification of Healthcare Resource Access Control (HRAC) Service. HRAC service is a mechanism for obtaining authorization decisions and administrating access control policies. It enables a common way for an application to request and receive an authorization decision. The service is intended to be used by security-aware applications.

2.2 Outstanding Issues

The following problems are specifically **NOT** addressed by this response:

2.2.1 Understanding of Application Functionality or Data

Intimate understanding of an application functionality may be needed for proper exercising of access control policies. The submission team refrained from making HRAC service interfaces syntax or semantics dependent or oriented towards functionality of any particular application or application domain. Instead, the design of the HRAC service introduces a notion of "secured resources" names and operations on them, as well as dynamic attributes and policy evaluators. This allows a more general approach that may be tailored to the specifics of most applications' functionality and the semantics of data they manipulate.

2.2.2 Format of Authorization Rules

The submission team decided NOT to specify interfaces through which authorization rules or policies are to be expressed. At the time of the initial submission, there was no any candidate for an IDL interface that the submission team would agree upon. The specified model of HRAC service allows policy evaluators that implement different authorization policies. Policy evaluators may have any interfaces and/or mechanism for expressing authorization rules governing access to secured resources. Such interfaces and/or mechanisms are beyond the scope of this submission.

2.2.3 Quality of Protection as Authorization Decision Factor

It may be reasonable to grant particular access to secured resources only if the quality of protection (QoP) for the reply is of sufficient strength (for example, data confidentiality and/or data integrity is guaranteed). QoP can be considered as another factor in authorization decisions. The current version of the submission is not providing mechanisms that allow QoP as a factor in authorization decisions. The submission team did not reach consensus on whether to provide such

mechanisms in HRAC specification or have a separate service (thus another specification) that would provide mechanisms for this. The team may address this problem in the revised submission.

2.2.4 Exceptions

In the current version of the submitted interface, exceptions are not present. It is not because the submission team believes that no exceptions should be raised by the corresponding objects. The exceptions will be specified in the future versions of the submission. At the time of this submission, we do not have consensus on what exceptions make sense to specify and how they would be used by programmers in developing ADO clients and HRAC services.

2.3 Design Goals

The submitters had the following goals in mind during the design of this submission:

2.3.1 Conservatism:

The proposal should extend the CORBAsecurity mechanisms rather than replacing them with a different model. The proposal should be implementable using CORBAsecurity as a base. Specifically, the proposal should use the CORBA security attribute structure to identify authenticated subjects to the access control mechanism.

2.3.2 Minimality:

The proposal should define the smallest number of interfaces and methods possible. The proposal should be easy to implement, and implementations should be small.

2.3.3 Simplicity:

The proposal should have a simple administrative model and a trivial runtime programming model.

2.3.4 Generality:

The proposal should be applicable to and useful in domains other than healthcare.

2.3.5 Relevance:

The proposal should satisfy the healthcare access control requirements set forth in the HRAC RFP. Specifically, the proposal should:

(a) define a notion of controlled resource, which allows extension of CORBA security protection to system entities other than CORBA objects.

(b) support enforcement of policies which take the following factors into account when making access control decisions:

- relationship between the requester and the accessed resource or its owner, subject, or referent
- value or sensitivity of information contained within resource
- time (e.g. time of day, day of week)

(c) support management of access control policy in a policy-language-independent way

(d) support OMG PIDS and COAS access scenarios

2.3.6 Flexibility:

The proposal should support a wide variety of policies (especially healthcare-appropriate policies). The proposal should be implementable using a variety of policy management and enforcement engines (including existing healthcare security packages).

2.3.7 Scalability:

The proposal should scale well, both in terms of runtime performance and in terms of management interface simplicity and management data size.

2.4 Reference Models

Two views of the HRAC are presented in the following models. The first is the access decision model. This represents the relationship of objects involved in making an access decision. The second view is the Administrative view and represents how an HRAC is configured. Administration of Access Policy is beyond the scope of the HRAC and is clearly indicated as such on this model diagram.

The Healthcare Resource Access control facility reference model defines a framework within which a wide variety of access control polices may be supported. The reference models below clearly indicate the scope of this submissions response by heavy dotted lines. In some cases there are types that occur within the scope of this response that represent concepts and/or services that lie beyond the scope of the HRAC. An example of this is the concept of a "secured resource" which is only represented within the scope of the HRAC by a ResourceName. Where this occurs these external concepts appear in the model, but outside the dotted line to aid the reader in an understanding of the relationship between the HRAC and the external concepts and/or services. The appearance of objects outside the scope of the submission is conceptual and is presented only to aid in understanding the types that occur within the HRAC.

HRAC types that represent or encapsulate external concepts and/or services:

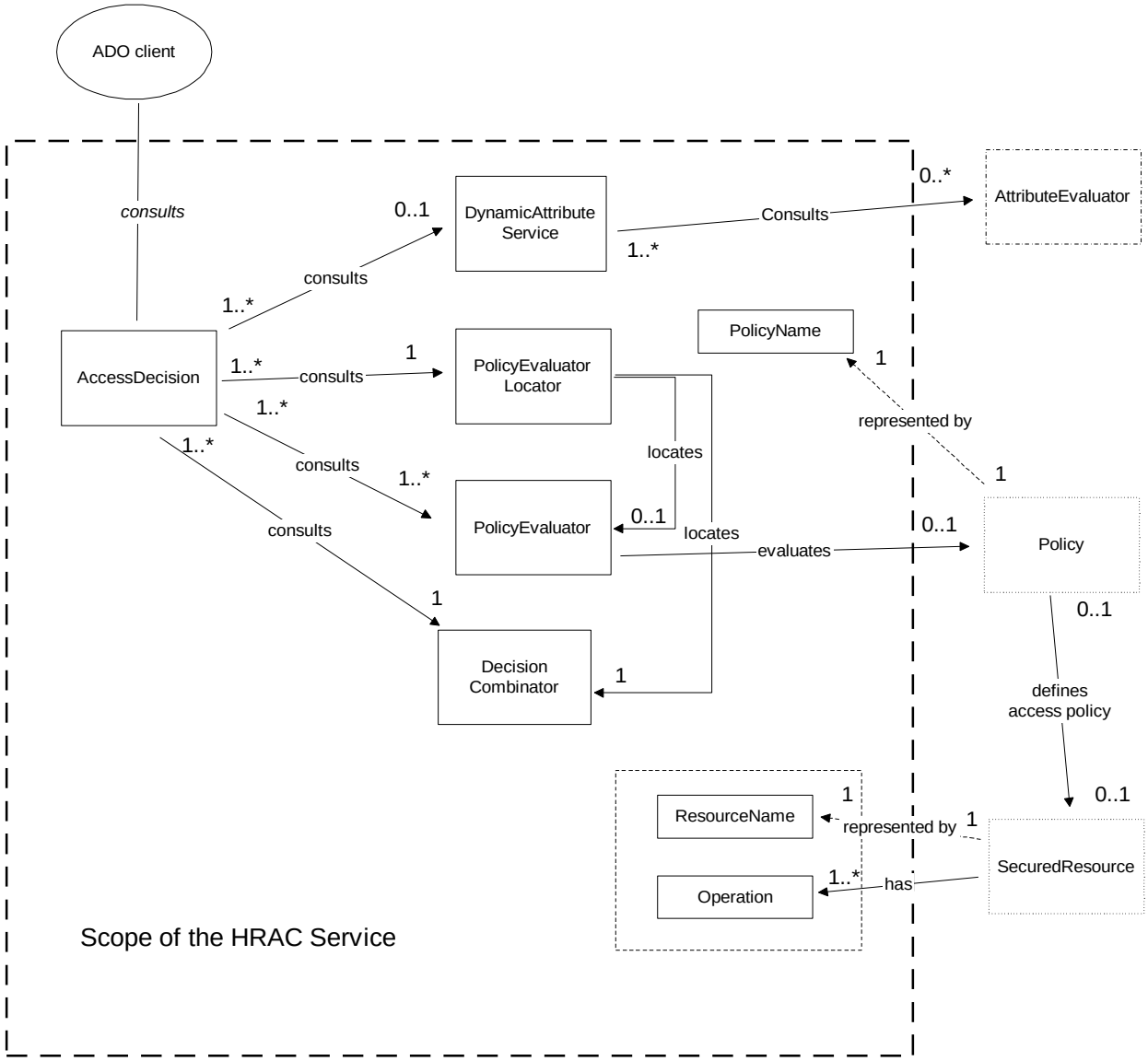
ResourceName: A "secured resource" is represented within the HRAC by a ResourceName that is a sequence of strings.

Operation: Secured resources have one or more operations which may be performed on them (such as create, get, set, use..). These operations are represented within the HRAC as strings.

PolicyName: "Policy" (the rules used for controlling access to secured resources and their operations) is beyond the scope of the HRAC, but when referenced within the HRAC, is identified by a PolicyName that is a string.

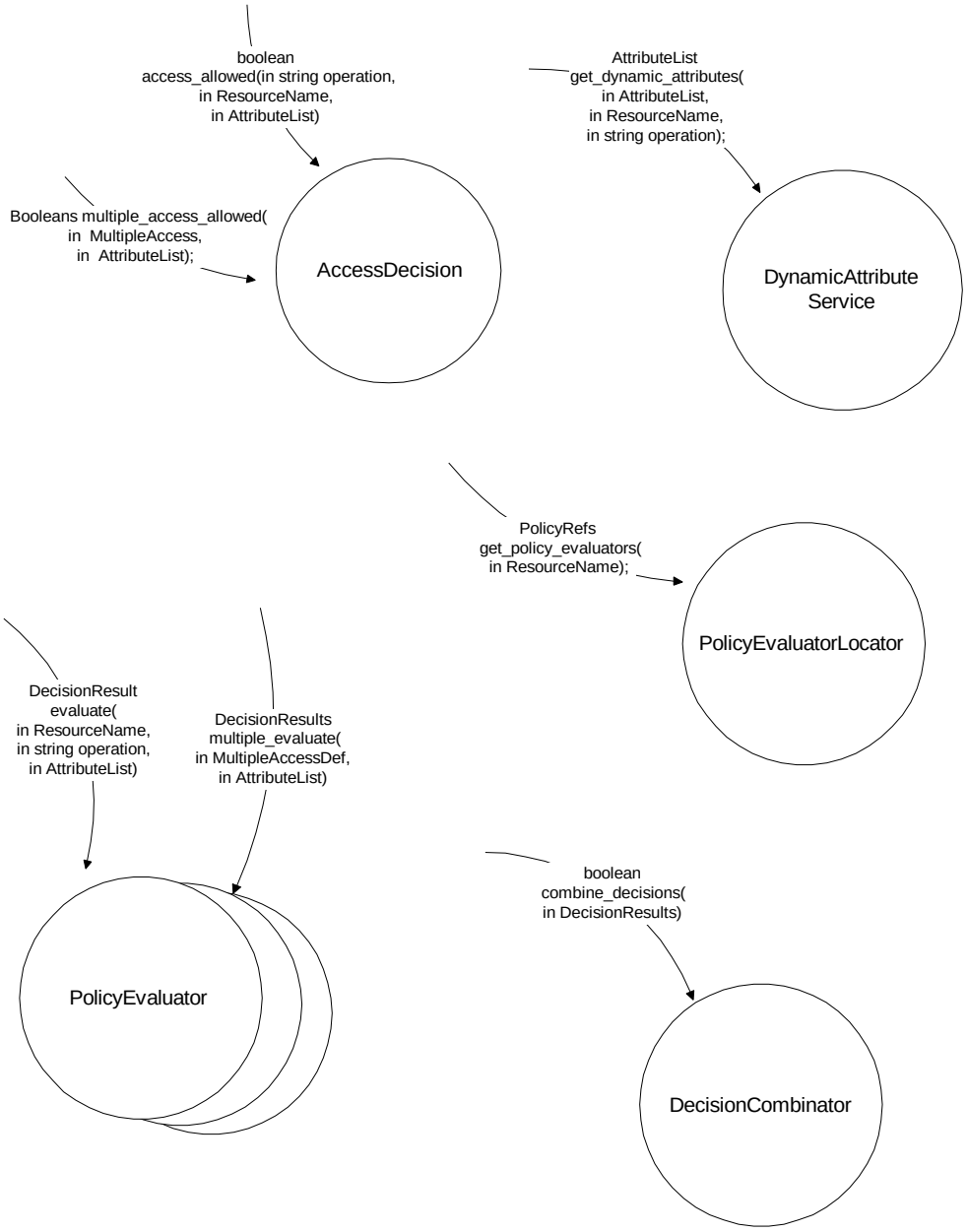
DynamicAttributeService: The DynamicAttributeService may consult an external AttributeEvaluator. The submitters plan to include standard administrative interfaces for this facility in the revised submission.

2.4.1 Access Decision Model



Healthcare Resource Access Control (HRAC)
Access Decision Model

An Access Decision is requested by a client by invoking the `access_allowed()` method of the `AccessDecision` object (ADO) passing a `ResourceName`, operation, and `SecAttributes`. The ADO consults a `DynamicAttributeEvaluator` to obtain an updated list of `SecAttributes` that include any dynamic attributes currently applicable for this access decision. The `DynamicAttributeService` may consult externally provided dynamic attribute evaluators as part of it's implementation. The `AccessDecision` object also consults the PolicyLocator to obtain object references for the `PolicyEvaluator`(s) and the `DecisionCombinator` that are required for an access decision. The AccessDecision object consults each `PolicyEvaluator`(s). PolicyEvalutators are responsible for interpreting access policy that controls access to the `ResourceName/operation`. The `AccessDecisionObject` consults (passes the results of those evaluate() methods) the DecisionCombinator who is responsible for understanding the policy that controls how a series of results from PolicyEvaluators are combined. It is the response from the `DecisionCombinator` that is returned to the client.

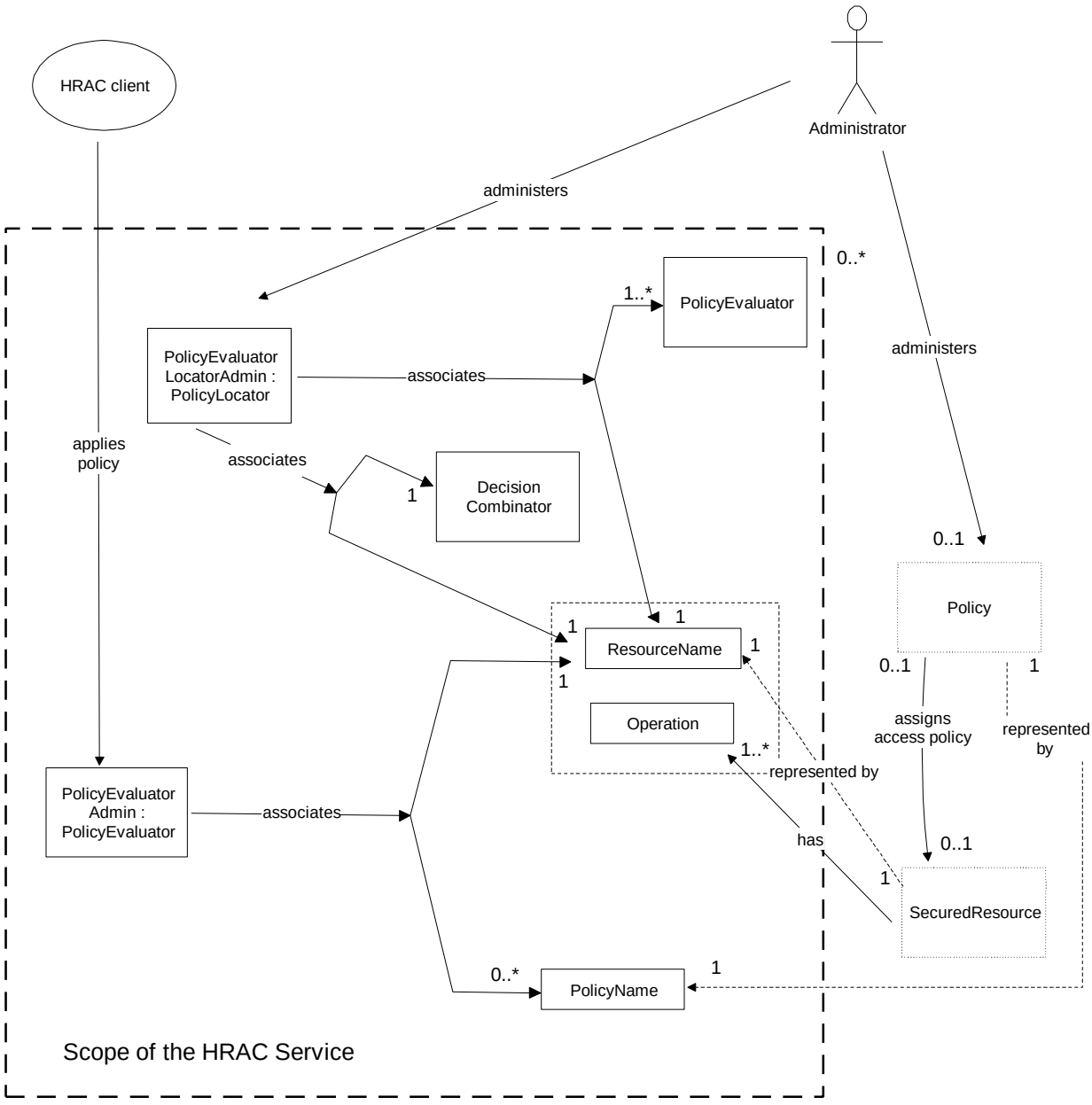


Interfaces involved in Access Decsions

10/13/98
9:04:08 PM
access_control_v3.vsd
Page 1 of 1

The interfaces and associated methods are described in detail in Section 3 of this submission.

2.4.2 Administrative Model



Healthcare Resource Access Control (HRAC)
Administrative Model

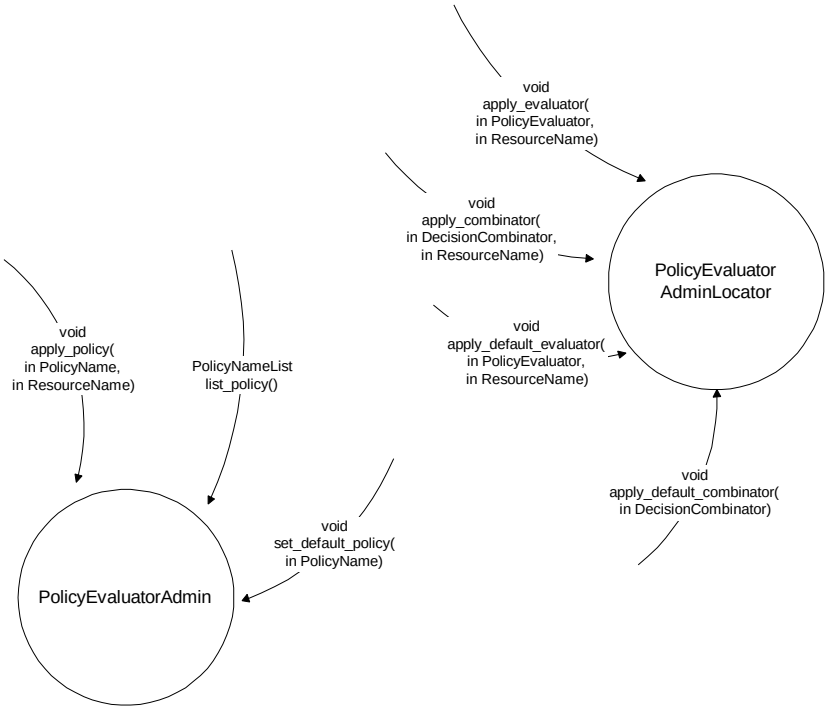
The administrative model of HRAC is designed to allow replaceable HRAC objects within an implementation and to allow HRAC clients to apply previously defined policy to resources.

The administrative model is **not** intended to provide the Administrative interfaces necessary to define access policy. The definition of access policy (the rules that govern access to secured resources/operations) is outside the scope of

this submission. This Administrative model clearly indicates this by placing Policy administration outside the dotted line that delineates the scope of the HRAC submission.

The PolicyEvaluatorLocatorAdmin interface is used to associate PolicyEvalutators and DecisionCombinators with a ResourceName. Multiple PolicyEvaluators may be associated with a single ResourceName. These evaluators will all be consulted during access decisions. There is only one DecisionCombinator provided for a ResourceName. This combinator is responsible for taking the results of the PolicyEvaluators evaluate() method and making a final access decision. PolicyEvaluators have an endless series of options for implementation. For this reason, the interface is public and evaluators may be "plugged-in" to an HRAC framework by vendors and/or users. In the same sense, there are many possible policies for combining policy decisions. Some secured resources should not be accessible unless all the PolicyEvaluators return ACCESS_DECISION_ALLOWED. Other secured resources may be accessible if any one of the PolicyEvaluators allow access. Defining an interface for the DecisionCombinators allows custom combinators to be configured for a secured resource. It is possible to configure a default DecisionCombinator.

The PolicyEvaluatorAdmin interface is used to apply an existing named access Policy to a secured resource. An application that wished to dynamically apply policy to newly created resources would be required to specify the names of those policies. The Policy would be configured by an administrator using the administrative interfaces of the underlying access policy system and the required name associated with it (this is outside the scope of the HRAC admin interfaces). Once this had been accomplished, an HRAC client could apply this named policy using the PolicyName to a ResourceName. The PolicyEvaluatorAdmin also allows default policy to be assigned "by name" and a list of existing PolicyNames can be retrieved from the object.



Interfaces involved in Administration

2.5 Discussion of Proposal Scope

2.5.1 Scope as defined by the RFP

The CORBAmed Healthcare Resource Access Control RFP defined the scope of proposals sought as follows:

> *"Mechanisms this RFP is asking for are sought to allow application systems to be unaware of advanced security policies existing in healthcare enterprises where those systems are deployed.*
>
> *This RFP scope is threefold:*
>
> 1. *to de-couple access control decision logic from application logic,*
> 2. *to provide a standard interface for the definition of access control rules,*
> 3. *to provide a standard interface for requesting access control decisions.*
>
> *An illustration of the RFP scope is provided in Figure 1. The RFP scope is limited to the additional security decision logic shown in the figure with striped background. It has a "Decision" interface to an interceptor(s) performing access control functions and an application itself to consult such Security Decision Logic for access control decisions. The "Admin" interface allows defining access control rules.*

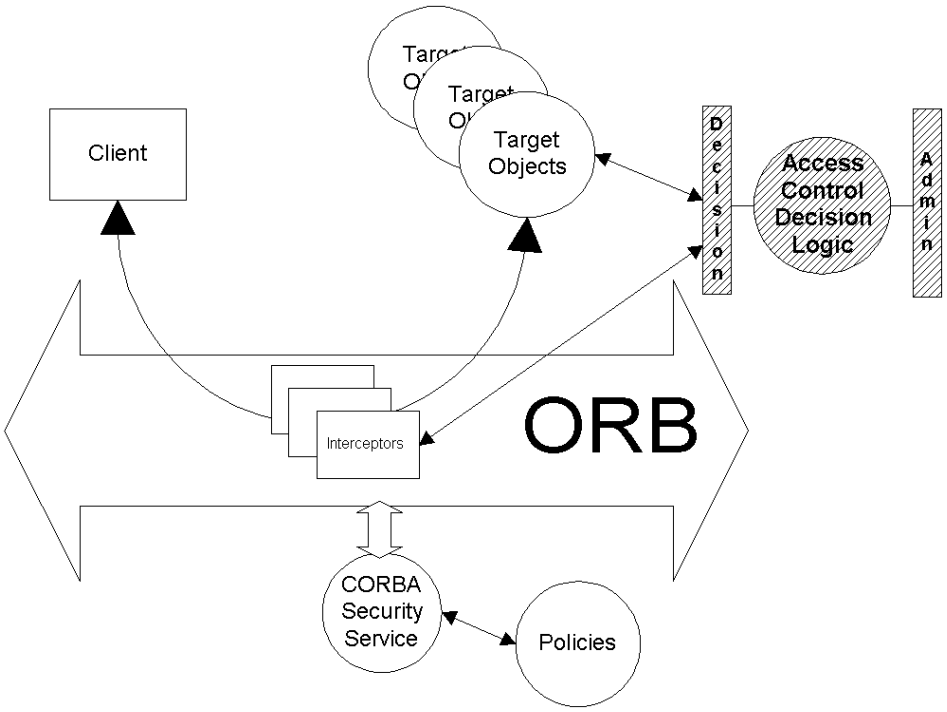


Figure 1: A Possible Solution

The RFP is looking for a solution where individual applications, or target objects, play the most minimal role possible in the realization of an enterprise security policy. Optimally, each

application's, or object's, contribution to security will be limited to requesting and enforcing access control decisions - without knowing or caring about how the decisions are made. This then allows the definition, implementation and management of each application, or object, and the enterprise-specific security policy to be orthogonal. "

2.5.2 The Scope of this submission

The initial submission addresses the following scope issues of the RFP

1. to de-couple access control decision logic from application logic

   The submission supports the separation of access control logic from the application. The submitters intend to extend this support again in the final submission by allowing applications to register dynamic attribute evaluators via a standard interface.

2. to provide a standard interface for requesting access control decisions.

   The AccessDecision interface provides this functionality.

In addition, this submission extends this scope to provide a frameword that supports the following:

1. replaceable authorization engines (PolicyEvaluatorLocator & PolicyEvaluatorLocatorAdmin, and PolicyEvaluator)
2. custom integration of multiple authorization engines (PolicyEvaluatorLocatorAdmin and DecisionCombinator)
3. use of dynamic attributes in access decisions (DynamicAttributeEvaluator)
4. The application of pre-defined access policy to a resource. (PolicyEvaluatorAdmin)

The submission does ***not*** address the following scope issue of the RFP:

1. to provide a standard interface for the definition of access control rules

   The submitters could not agree on IDL for the definition of access control rules. This is primarily because there are so many diverse ways that people express access control policies and accomodation from a single IDL interface for this diversity is not an easy task. The final submission may include an example of how the CORBAsecurity required rights model could be used to provide for the definition of access control rules **if** access control policy uses a "required rights" model. In general, the administration of access control policy was felt to be out of scope of this submission.

3. DfResourceAccessControl module

```
//File: DfResoureAccessControl

#ifndef _DF_RESOURCE_ACCESS_CONTROL_IDL_
#define _DF_RESOURCE_ACCESS_CONTROL_IDL_

#include <orb.idl>
#include <Security.idl>

#pragma prefix "omg.org"

module DfResourceAccessControl
{

interface AccessDecision {
…
};

interface DynamicAttributeService {
…
};

interface PolicyEvaluatorLocator {
…
};

interface PolicyEvaluatorLocatorAdmin {
…
};

interface PolicyEvaluator {
…
};

interface PolicyEvaluatorAdmin {
…
};

interface DecisionCombinator {
…
};

};

#endif // _DF_RESOURCE_ACCESS_CONTROL_IDL_
```

The DfResourceAccessControl contains four interfaces defined below and has type dependencies on the CORBA Security Service and the CORBAMed NamingAuthority modules.

`#include <Security.idl>`

The types declared within the Security service and used by the HRAC are:

`Security::AttributeList`

These types are used for consistency with CORBASec and have the same meaning when used in HRAC interfaces. They are typedef'd in this specification for ease of use.

#pragma prefix "omg.org"

In order to prevent name pollution and name clashing of IDL types this module (and all modules defined in this specification) uses the pragma prefix that is the omg DNS name.

3.1 Types

There are a number of structured types used widely through out the DfResourceAccessControl Model. These types are described in this section:

3.1.1 Basic Types & Types used from the CORBA Security service

```
//*********************************************************
//         Basic Types
//*********************************************************

typedef sequence<boolean> BooleanList;

typedef Security::AttributeList AttributeList;
```

BooleanList

A sequence of boolean used as a return value when multiple decisions are requested. This type is used as a return value in the multiple_access_allowed() method of the AccessDecision interface.

AttributeList

The Security::AttributeList is defined as follows in CORBA Security 1.2 (ptc/98-01-02). The AttributeList is provided as an input parameter by the "application" client when a request for an access decision is made. The AttributeList used for access decisions may be modified to include dynamic attributes by use of the get_dynamic_attributes() method of the DynamicAttributeService interface. As a convenience to the reader, the structure of a Security::AttributeList is replicated below.

```
typedef sequence<octet> Opaque;

// security attributes
typedef unsigned long SecurityAttributeType;


struct ExtensibleFamily {
    unsigned short        family_definer;
    unsigned short        family;
};
struct AttributeType {
    ExtensibleFamily      attribute_family;
    SecurityAttributeType attribute_type;
};


struct SecAttribute {
    AttributeType         attribute_type;
    Opaque                defining_authority;
    Opaque                value;
    // the value of this attribute can be
    // interpreted only with knowledge of type
};
```

```
typedef sequence <SecAttribute> AttributeList;
```

3.1.2 Types that identify and manage information about secured resources

```
//*******************************************************
//   Types that identify a secured resource
//*******************************************************

typedef sequence<string> ResourceName;

typedef sequence<string> OperationList;
```

`ResourceName`

A ResourceName is used to identify a **secured resource**. ResourceName is a sequence of string allowing for groupings of resources. It is required that the first string in the sequence is formatted as a NamingAuthority::QualifiedNameStr. This ensures globally unique resource names (and/or groups) See NamingAuthority module in corbamed/98-02-29. Issue: Submitters are discussing whether we want to make this a stronger requirement by putting the type in the IDL (this is a typedef for a string with a standard format in NamingAuthority).

`OperationList`

An OperationList is used to identify a list of operations that may be performed on a secured resource.

3.1.3 Types associated with Administering Access Policy

```
//*************************************************
//   Types associated with Administering Access Policy
//*************************************************
typedef string PolicyName;
typedef sequence<PolicyName> PolicyNameList;

const PolicyName NO_ACCESS_POLICY = "NO_ACCESS_POLICY";

interface PolicyEvaluator;
typedef sequence<PolicyEvaluator> PolicyEvaluatorList;

struct PolicyDecisionEvaluators {
    PolicyEvaluatorList    policy_evaluator_list;
    DecisionCombinator    decision_combinator;
};
```

`PolicyName`

A PolicyName is a string used to identify an access policy for a secured resource. This type is as an input parameter to the apply_policy(), and set_default_policy() methods of the PolicyEvaluatorAdmin interface. PolicyNames are assigned by the administrative interface of the policy engine and cannot be modified or control by the HRAC. There is one standard PolicyName of "NO_ACCESS_POLICY" that may be used to remove policy from a secured resource.

PolicyNameList

A PolicyNameList is a sequence of PolicyNames. It is returned from the list_policy() method of the PolicyEvaluatorAdmin interface.

PolicyEvaluatorList

A PolicyEvaluatorList is a sequence of PolicyEvaluator object references.

PolicyDecisionEvaluators

The PolicyDecisionEvaluators struct contains a sequence of PolicyEvaluator object references and the DecisionCombinator returned from the get_policy_decision_evaluators() method of the PolicyEvaluatorLocator interface. This structure contains the references of all the objects that must be consulted during an access decision.

3.1.4 Types related to Resource Access Decision Rules

```
//*************************************************
//      Types used to request an Access Decision
//*************************************************

struct AccessDef {
    ResourceName   resource_name;
    string         operation;
};
typedef sequence<AccessDef> AccessDefList;

enum DecisionResult {ACCESS_DECISION_ALLOWED,
                     ACCESS_DECISION_NOT_ALLOWED,
                     ACCESS_DECISION_UNKNOWN
};

typedef sequence<DecisionResult> DecisionResultList;
```

AccessDef

The AccessDef struct is provided to allow multiple access definitions to be defined. It contains the ResourceName and the operation name for the secured resource access being requested.

AccessDefList

AccessDefList is the type used to request multiple access decisions in a single operation. It is used as an input parameter to the multiple_access_allowed() method of the AccessDecision interface and the multiple_evaluate() method of the PolicyEvaluator interface.

DecisionResult

This type is used as a result in access decisions where access policy is applied. This is the type returned from the evaluate() method of the PolicyEvaluator.

The meanings are:

ACCESS_DECISION_ALLOWED: the policy evaluated for this ResourceName, operation and Attribute list indicates that access is ALLOWED.

ACCESS_DECISION_NOT_ALLOWED: the policy evaluated for this ResourceName, operation and Attribute list indicates access is NOT_ALLOWED.

ACCESS_DECISION_UNKNOWN: the policy evaluated for this ResourceName, operation and Attribute list indicates an access decision cannot be made.

DecisionResultList

DecisionResultList is a sequence of DecisionResult. This is the type returned from the multiple_evaluate() method of the PolicyEvaluator and is the type provided as an input parameter to the combine_decisions() method of the DecisionCombinator.

3.1.5 Exceptions

The following exceptions are generally useful by most or all of the interfaces of this module.

```
Exceptions will be added in the final submission.
```

3.2 AccessDecision interface

```
//*************************************************
//     interface AccessDecision
//*************************************************

interface AccessDecision {

    boolean access_allowed(
        in ResourceName    resource_name,
        in  string         operation,
        in  AttributeList  attribute_list
    );

    BooleanList multiple_access_allowed(
        in  AccessDefList      access_def_list,
        in  AttributeList       attribute_list
    );

};
```

The Access Decision object is used to request decisions on access based on a ResourceName, an Operation, and a list of SecAttributes. This submission provides a framework for the support of many policy evaluators. It is out of the scope of this submission to mandate how policy is defined or evaluated using the information provided by the client at the time access decisions are requested. This is the only interface that is necessary for a client to be familiar with in order to obtain access decisions from the HRAC.

access_allowed()

A single access decision is requested and a boolean is returned

multiple_access_allowed()

Multiple access decisions are requested in a single method invocation and a sequence of booleans are returned. The boolean sequence maps one to one in the same order to the provided sequence of ResourceName/operation pairs.

3.3 DynamicAttributeService interface

```
//*****************************************************
//     interface DynamicAttributeService
//*****************************************************

interface DynamicAttributeService {

    AttributeList get_dynamic_attributes(
        in  AttributeList   attribute_list,
        in  ResourceName    resource_name,
        in  string          operation
    );
};
```

The DynamicAttribute interface is used to obtain a new list of SecAttributes that are applicable to an access decision. This service may encapsulate calls to a relationship service and/or application specific logic to determine how the original AttributeList provided by the client should be modified.

NOTE: It is the intent of the submitters to provide an administrative interface for the DynamicAttributeService in the final submission. This would provide a standard way for an application to register custom DynamicAttributeService(s) that would be used by this object at access decision time to determine applicable dynamic attributes.

get_dynamic_attributes()

This method takes the parameters provided by the client of the AccessDecision object; the AttributeList, the ResourceName, and the operation and determines what (if any) dynamic attributes should be added to the AttributeList. In addition, the returned AttributeList may be modified by this service. The service may add or remove SecAttributes to this list. It is the returned list of SecAttributes that is used as the basis of access decisions by the HRAC.

3.4 PolicyEvaluatorLocator interface

```
//*****************************************************
//     interface PolicyEvaluatorLocator
//*****************************************************

interface PolicyEvaluatorLocator {

    readonly attribute PolicyEvaluatorLocatorAdmin policy_evaluator_locator_admin;

    PolicyDecisionEvaluators get_policy_decision_evaluators(
        in  ResourceName    resource_name
    );

};
```

The PolicyEvaluatorLocator interface is used to locate the PolicyEvaluators and the DecisionCombinator associated with a ResourceName. This submission provides a framework for the support of one or more policy evaluators for a single resource.

readonly attribute PolicyEvaluatorLocatorAdmin

If the PolicyEvaluatorLocator has an associated administrative interface, it can be obtained via this attribute. If an administrative interface is not available for this evaluator, this attribute will be nil.

get_policy_decision_evaluators()

A PolicyDecisionEvaluators structure which contains a list of PolicyEvaluator object references and the DecisionCombinator object reference for the resource is returned to the client.

3.5 PolicyEvaluatorLocatorAdmin interface

```
//*****************************************************
//     interface PolicyEvaluatorLocatorAdmin
//*****************************************************

interface PolicyEvaluatorLocatorAdmin {

    void add_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in  ResourceName resource_name
    );

    void replace_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in   ResourceName resource_name
    );

    void set_default_evaluators(
        in  PolicyEvaluatorList policy_evaluator_list
    );

    void apply_combinator (
        in  DecisionCombinator decision_combinator,
        in  ResourceName   resource_name
    );

    void set_default_combinator(
        in  DecisionCombinator decision_combinator
    );

};
```

The PolicyEvaluatorLocatorAdmin object is used to associate PolicyEvaluators with a ResourceName. The object is also used to associate the appropriate DecisionCombinator with the ResourceName. This submission provides a framework for the support of one or more policy evaluators for a single resource.

add_evaluators()

A list of PolicyEvaluators is added to the list of evaluators for the named resource. These evaluators will be in the list of PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method. The addition of evaluators to a ResourceName which previously had none results in the added list of evaluators being the only evaluators consulted on an access decision (system default evaluators are no longer consulted).

replace_evaluators()

A list of PolicyEvaluators is assigned for the named resource. If the resource had existing PolicyEvaluators assigned, they are removed and the entire list is replaced with the ones provided in this method. These evaluators will be the PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

set_default_evaluators()

The list of PolicyEvaluators provided is set as the default evaluators for any ResourceName for which PolicyEvaluators have not been explicitly assigned. Default evaluators are overridden by the add_evaluators() or replace_evaluators() methods. The default evaluators will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method when no PolicyEvaluators have been explicitly assigned for a ResourceName. Additional PolicyEvaluators may be added with the add_evaluators() method of this interface, or the defaults replaced with the replace_evaluators() method of this interface.

apply_combinator()

A DecisionCombinator is specified for the named resource. This combinator will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method. The DecisionCombinator provided replaces any previous combinator specified for the secured resource.

set_default_combinator()

The DecisionCombinator provided is set as a default. This combinator is now the combinator used when a DecisionCombinator has not been explicitly applied for a secured resource. This combinator will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method for these resources.

3.6 PolicyEvaluator interface

```
//*****************************************************
//     interface PolicyEvalutator
//*****************************************************

interface PolicyEvaluator {

    readonly attribute PolicyEvaluatorAdmin policy_evaluator_admin;

    DecisionResult evaluate(
        in  ResourceName   resource_name,
        in  string         operation,
        in  AttributeList  attribute_list
    );

    DecisionResultList multiple_evaluate(
        in  AccessDefList  access_def_list,
        in  AttributeList  attribute_list
    );

};
```

The PolicyEvaluator interface is used to obtain an access decision based on an encapsulated policy for the ResourceName/operation when provided a list of effective Security Attributes for the requestor. This submission provides a framework for the support of one or more policy evaluators for a single resource.

readonly attribute PolicyEvaluatorAdmin

If the PolicyEvaluator has an associated administrative interface, it can be obtained via this attribute. If an administrative interface is not available for this evaluator, this attribute will be nil.

evaluate()

A single access decision is requested based on access policy(s) this evaluator determines is appropriate for the named resource. The decision is based on the ResourceName, the operation, and the effective Security Attributes. The SecAttributes passed to the AccessDecision object by the client in access_allowed() may have been modified by the DynamicAttributeService get_dynamic_attributes() method before the PolicyEvaluator is called. The DecisionResult is a ternary result. The DecisionResult is as follows:

ACCESS_DECISION_ALLOWED: the policy evaluated for this ResourceName, operation and Attribute list indicates that access is ALLOWED.

ACCESS_DECISION_NOT_ALLOWED: the policy evaluated for this ResourceName, operation and Attribute list indicates access is NOT_ALLOWED.

ACCESS_DECISION_UNKNOWN: the policy evaluated for this ResourceName, operation and Attribute list indicates an access decision cannot be made.

multiple_evaluate()

A multiple access decision is requested based on access policy(s) this evaluator determines is appropriate for the named resources. Each decision is based on the ResourceName, the operation, and the effective Security Attributes. The SecAttributes passed to the AccessDecision object by the client in access_allowed() may have been modified by the DynamicAttributeService get_dynamic_attributes() method before the PolicyEvaluator is called. The DecisionResults is a sequence of ternary result as defined in the evaluate() method. The DecisionResults sequence maps one to one in the same order to the provided sequence of ResourceName/operation pairs.

3.7 PolicyEvaluatorAdmin interface

```
//*****************************************************
//     interface PolicyEvalutatorAdmin
//*****************************************************

interface PolicyEvaluatorAdmin {

    void    replace_policy(
        in  PolicyName  policy_name,
        in  ResourceName resource_name
    );

    void    add_policy(
        in  PolicyName  policy_name,
        in  ResourceName resource_name
    );

    PolicyNameList list_policy();

    void    set_default_policy(
        in  PolicyName  policy_name
    );
};
```

The PolicyEvaluatorAdmin interface is used to associate named access policies with secured resources. It is assumed that the administrative tool used to create and manage access policies (outside the scope of this submission) provides a mechanism to allow policies to be associated

with “names” which are represented as PolicyName (a string). This PolicyEvaluatorAdmin interface allows those policies to be applied “by name” to a secured resource represented by a ResourceName.

This interface is primarily provided for the application that wishes to assign a policy to a newly created resource programatically at the time of resource creation. It does, however, require that the application have knowledge of the named policies in order to choose an appropriate policy for access decisions.

`replace_policy()`

The policy identified by PolicyName is associated with the secured resource identified by the ResourceName. If the PolicyName is NO_ACCESS_POLICY, then all policy is removed for the resource. If a PolicyName is applied to a resource that has existing policy, then the policy will be replaced by the policy identified by the PolicyName.

`add_policy()`

The policy identified by PolicyName is added to the list of policies used when making access decisions for the secured resource identified by the ResourceName. If a PolicyName is added to a resource that has existing policy, then the policy will be added to the list of policies that control access decisions for the resource. An implementation is not required to support multiple policies for a resource. If the implementation does not support the application of multiple policies, then an exception (ISSUE: which one) shall be thrown for this method.

`list_policy()`

A list of all existing PolicyNames is returned to the client.

`set_default_policy()`

This method sets a default policy for any secured resource which has not yet been assigned an access policy.

3.8 DecisionCombinator interface

```
//*******************************************************
//     interface DecisionCombinator
//*******************************************************

interface DecisionCombinator{

    boolean combine_decisions(
        in DecisionResultList decision_result_list
    );
};
```

The DecisionCombinator interface is used to combine the decisions of multiple PolicyEvaluators. Combinators may be provided with different behaviors. A combinator that supported an “ANY” policy would return TRUE if any of PolicyEvaluators returned ACCESS_DECISION_ALLOWED. A combinator that supported an “ALL” policy would return TRUE only if all of PolicyEvaluators returned ACCESS_DECISION_ALLOWED. DecisionCombinators might also be arbitrarily complex A default combinator may be used for all access decisions, or combinators may be chosen specifically for access decisions on specific secured resources. The submitters have agreed that complex combinators may require more information than what is currently passed in the DecisionResultList. It is, however, still an issue

of exactly what would be useful in the general case as additional parameters to the DecisionCombinator. This issue will be address prior to the final submission.

`combine_decisions()`

The DecisionCombinator takes the DecisionResults from all of the PolicyEvaluators and returns a boolean result. This is the result that will be returned by the AccessDecision object to the original client of the HRAC facility.

4. DfHealthCareResource module

This module will be added in the final submission and will contain the healthcare specific secured resources requested in the RFP. The submitters are planning to work with the Clincial Observation Access Service (COAS) submitters to ensure that this list of resources is compatible with the requirements of the COAS submission.

5. Conformance Classes

Conformance classes will be defined in the final submission.

6. Appendix - Use Case Examples

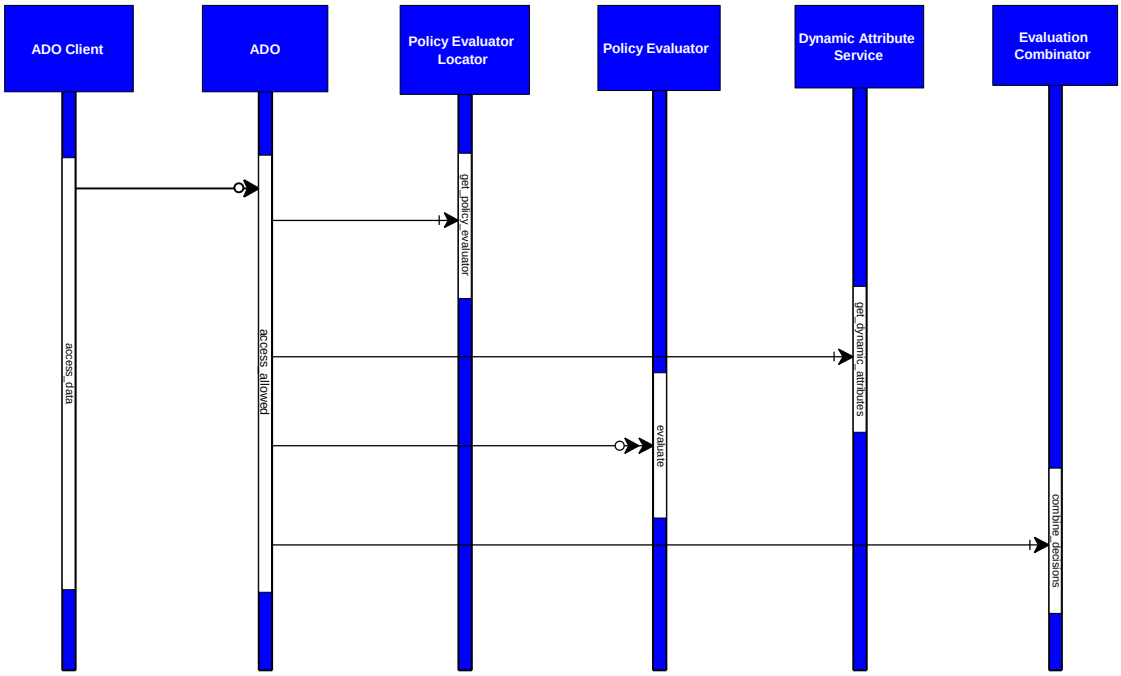
This appendix presents examples illustrating healthcare scenarios and the use of HRAC to provide access control for the instances of healthcare information access implied by these scenarios. Each example consists of several Use Cases:

1. A description of the healthcare scenario which involves one or more accesses to healthcare information.
2. For each healthcare information access required by the healthcare scenario:
   - A description of the actions of the healthcare application, the client of the Access Decision Object (ADO).
   - A description of ADO actions.
   - An Object Interaction Diagram (OID) for the ADO showing the arguments passed and the returns for each object invocation.

Before presenting the Use Cases, a generic Object Interaction Diagram (OID) describing the ADO is provided.

6.1 Generic ADO Object Interaction Diagram

This section shows the generic Object Interaction Diagram for the ADO.



6.2 Healthcare Scenario: Out-patient Visit to Attending Physician

This scenario (see table 1) illustrates the interaction with a patient record as a result of a patient's visit with an attending physician at the hospital on an out-patient basis. In this example, the access control policy pertinent to this scenario is called the "Basic Hospital Patient Record Access Policy."

As described in more detail in the normative part of this document, a policy within HRAC consists of an evaluator applied to static attributes, dynamic attributes, and other factors, such as, time of day and location of the principal. An evaluator can be implemented as an interpreter of rules expressed in some scripting language, e.g., SQL, as a process for which the rules are encapsulated as part of the process, e.g., Java Classes, or as some combination of these methods.

Static attributes are used for describing relatively fixed properties of users and resources, such as, basic user role and resource creation date. The values of static attributes are typically set by a security administrator and are obtained by the application in an implementation specific manner, e.g., from the principal's credentials. While the use of a static attribute in policy is specified by a security administrator, the values of dynamic attributes are typically set as part of normal information processing. Unlike static attributes which are usually properties of (i.e.,metadata about) information content, values of dynamic attributes are information content which are necessary to make an access decision. Some examples of dynamic attributes, which may be contained in a patient record or elsewhere, are:

A list of physicians, i.e., attending physicians, which are currently treating the patient.

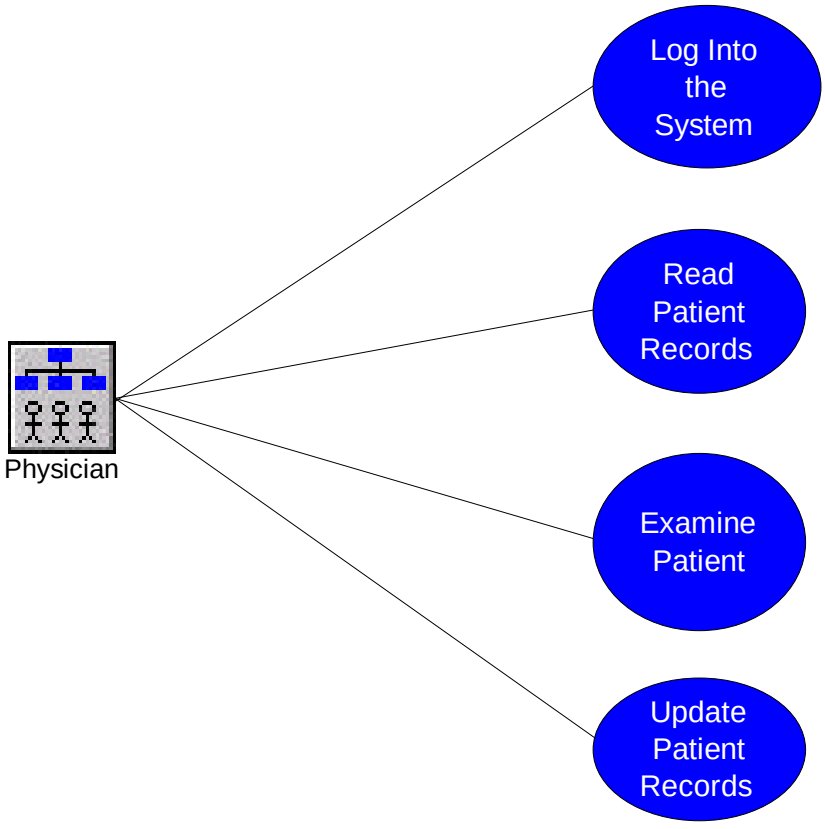
An authorization permitting the release of mental health information to designated parties.

Depending on the implementation, a dynamic attribute may be the value of the dynamic attribute or a reference to the value of the dynamic attribute. If a reference, then the dynamic attribute value is obtained by the evaluator if and when the evaluator determines that the value is needed to make the access decision.

HRAC is able to support more than one access policy. This healthcare scenario describes HRAC functionality using the Basic Hospital Patient Record Access Policy. Different access policy evaluators may be implemented by different developers. Dynamic attributes may be associated with only one or several evaluators. New dynamic attributes may be added to the Dynamic Attribute Service of an HRAC when new evaluators are installed. Once dynamic attributes are added to the Dynamic Attribute Service, they may be available for use by all evaluators. In addition to the Basic Hospital Patient Record Access Policy, other policies may specify access control requirements for HIV or mental health information resources which are part of the patient record.

The Basic Hospital Patient Record Access Policy used in this example specifies the conditions under which an attending physician can access a patient record. The policy specifies that attending physicians may read/update a patient record and/or modify certain authorization settings in a patient record. Within this policy, the term "update" when applied to clinical information refers to an append operation. Clinical information in the patient record once entered may not be modified.

Several static and dynamic attributes are used by the HRAC evaluator which implements the Basic Hospital Patient Record Access Policy. Among these are the static attribute "role" and the dynamic attribute "principal/patient_relationship." The value of the static attribute role specifies the basic role of a user, such as, physician, nurse, registrar. In this example, the value of role is obtained from the principal's credentials. The value of the dynamic attribute principal/patient_relationship specifies the relationship between the principal accessing the patient record and the patient who is the subject of the patient record being accessed, e.g., "primary_care," "attending," "consulting." In this example, the value of the principal/patient_relationship dynamic attribute is obtained by the Dynamic Attribute Service by accessing the content of the patient record which contains a list of attending physicians.

Use Case Name		Out-patent Visit to Attending Physician
Goal in Context		Physician provides care to a visiting patient
Scope & Level		Summary
Preconditions		Patient records already exist in the system, there is already some kind of relationship between the patient and the physician (attending, consulting, admitting, etc.)
Success End Condition		Patient records are updated according to the visit results.
Failed End Condition		Patient records are not updated according to the visit results.
Primary Actors		Care providing physician
Secondary Actors		
Trigger		Patient visits corresponding physician.
Applicable Access Policy		Basic Hospital Patient Record Access
Diagram		
Description	Step	Action

	1	Physician (or physician representative) logs into the information system unless it was done previously.
	2	Physician retrieves patient records and browses them.
	3	Physician examines the patient.
	4	Physician updates patient records.
Extensions	Step	Branching Action
	4 a	Physician changes authorization settings for the patient records (or their sub-set) according to the patient request and/or sensitivity of the information with which records are updated.
Variations	Step	Branching Action
		No variations
Related Information		
Priority		High
Performance		1 hour
Frequency		Many times per hour through the hospital
Channels to actors		Vision, speech, various instruments and devices in order to examine the patient; computer GUI to log into the system, brows and update patient records.
Open Issues		What authorization settings of the patient records can a related physician change?
		What if another related physician has limited access to records that are interesting in the context of the visit and the patient agrees those records can be disclosed?
Superordinate use cases		No superordinates
Subordinate use cases		Log into the system, Read Patient Records, Examine Patient, Update Patient Records, Change Authorization Settings for the Patient Record(s).

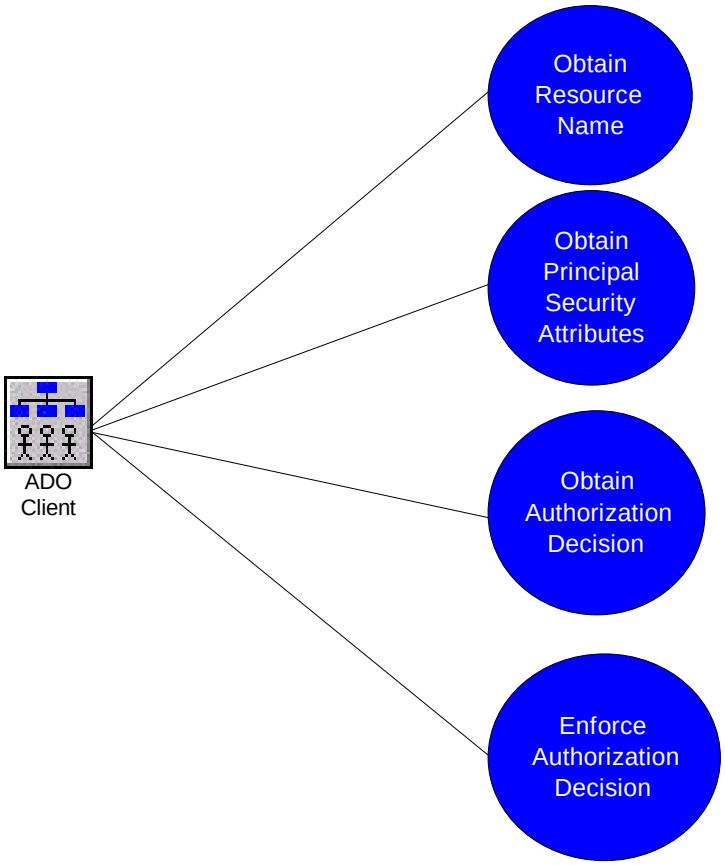
Table 1: Healthcare Scenario: Out-patient Visit to Attending Physician

As shown in table 1, there are three types of access to the patient record involved in this scenario: read, update, and change authorization.

The next section describes the actions of the application program (the ADO client) in reading the patient record including how the ADO is used to determine access according to the Basic Hospital Patient Record Access Policy.

6.2.1 ADO Client Actions: Read Patient Record

Use Case Name	ADO Client Actions: Read Patient Record
Goal in Context	Application program (ADO client) browses patient record.
Scope & Level	Subfunction
Preconditions	Patient records already exist in the system; physician has logged into application program; application program initiated successfully.
Success End Condition	The intended part of patient records are "read" accessed by the caregiver.
Failed End Condition	The intended part of patient records are not "read" accessed by the caregiver.
Primary Actors	1. Client program acting on behalf of the caregiver (Client) 2. CORBA-compliant application service (Service), which provides "read" access to the required information
Secondary Actors	1. Access Decision Object (ADO), which provides interface DfResourceAccessControl::AccessDecision
Trigger	A caregiver is attempting to "browse" parts of the patient medical record.
Applicable Access Policy	Basic Hospital Patient Record Access: An attending physician may read any part of the patient record.

Diagram		
Description	Step	Action
	1	Application program (ADO client), acting on behalf of the physician, obtains the resource_name for the part of the patient record to be read and static_attributes.
	2	ADO client invokes access_allowed(resource_name, "read", static_attributes).
	3	If access_allowed() returns "true," then ADO client reads and displays requested part of the patient record to physician; otherwise, ADO Client displays error.
Extensions	Step	Branching Action
		No variations
Variations	Step	Branching Action
		No variations
Related Information		
Priority		High
Performance		
Frequency		Many times per hour through the hospital
Channels to actors		
Open Issues		
Superordinate use cases		Out-patent Visit to Attending Physician

Subordinate use cases	ADO Actions: Read Patient Record

Table 2: ADO Client Actions: Read Patient Record

Table 2 describes the actions of the application program (ADO client) in providing the physician the capability of browsing resources contained in the patient record. The application program obtains from the physician the name of the resource to be read. It then obtains the static attributes from the physician's credentials. The application invokes the ADO which returns an indication of whether the physician is able to read the requested resource within the patient record. If the physician has read access to the resource, the application displays the resource for the physician.

The next section describes the actions of the ADO when it is invoked by the application to determine if the physician has read access to the patient record resource.

6.2.2 ADO Actions: Read Patient Record

Use Case Name	ADO Actions: Read Patient Record
Goal in Context	ADO renders access decision for a resource which is part of the patient record.
Scope & Level	Subfunction
Preconditions	Patient records already exist in the system; Application program has invoked ADO.
Success End Condition	An access decision is returned by the ADO to the application program.
Failed End Condition	An exception occurred and an access decision is not returned by the ADO to the application program.
Primary Actors	1. Access Decision Object (ADO), which provides interface DfResourceAccessControl::AccessDecision
Secondary Actors	1. Policy Locator Object(PL), which provides the interface DfResourceAccessControl::PolicyEvaluatorLocator 2. Dynamic Attribute Service Object(DAS), which provides interface DfResourceAccessControl::DynamicAttributeService 3. Policy Evaluator Object (PE), which provides the interface DfResourceAccessControl::PolicyEvaluator 4. Decision Combinator Object(PCO), which provides the interface DfResourceAccessControl::DecisionCombinator
Trigger	Application program (ADO client) invokes ADO.
Applicable Access Policy	Basic Hospital Patient Record Access: An attending physician may read any part of the patient record

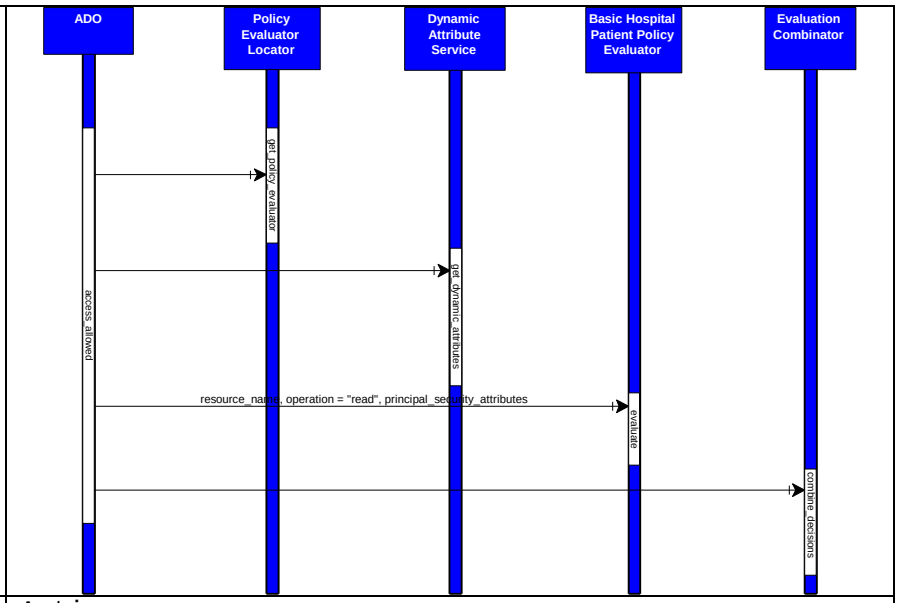
Diagram		
Description	Step	Action
	1	ADO invokes get_policy_evaluator(resource_name)which returns Policy_Evaluator for the Basic Hospital Patient Record Access Policy.
	2	ADO invokes get_dynamic_attributes(static_attributes, resource_name, operation) which returns attributes', a list of all static and dynamic attributes required for Policy_Evaluator to make the access decision.
	3	ADO invokes Policy_Evaluator.evaluate(resource_name, attributes', "read") which returns trinary_result.
	4	ADO returns the boolean result from combine_decisions({trinary_result})
Extensions	Step	Branching Action
		No variations
Variations	Step	Branching Action
		No variations
Related Information		
Priority		High
Performance		
Frequency		Many times per hour through the hospital
Channels to actors		
Open Issues		
Superordinate use cases		ADO Client Actions: Read Patient Record
Subordinate use cases		

Table 3: ADO Actions: Read Patient Record

Table 3 describes the actions of the ADO in providing an access decision when invoked by the application in order to determine if the physician the capability of browsing resources contained in the patient record. Given resource_name, a resource within the patient record, the operation "read," and static_attributes, a list of static attributes, the ADO invokes get_policy_decision_evaluators() with the resource_name which returns the single policy evaluator, Policy_Evaluator, for the Basic Hospital Patient Record Access Policy. The ADO obtains dynamic attributes by invoking get_dynamic_attributes() with static_attributes, resource_name, and the operation "read." A combined list of static and dynamic attributes is now contained in attributes'. The ADO then invokes the evaluator referenced by Policy_Evaluator which returns a trinary result, i.e., ACCESS_DECISION_ALLOWED, ACCESS_DECISION_NOT_ALLOWED, ACCESS_DECISION_UNKNOWN. If attributes' contains both the static attribute "physician" and the dynamic attribute "attending," then the result from policy_evaluator is ACCESS_DECISION_ALLOWED in accordance with the Basic Hospital Patient Record Access Policy. Finally, the ADO invokes combine_decisions() with Policy_Evaluator and the result from the invocation Policy_Evaluator and returns the result from combine_decisions() to the application.

7. Appendix - Complete IDL

```
//File: DfResourceAccessControl.idl
//

#ifndef _DF_RESOURCE_ACCESS_CONTROL_IDL_
#define _DF_RESOURCE_ACCESS_CONTROL_IDL_

#include <orb.idl>

#include "Security.idl"

#pragma prefix "omg.org"

module DfResourceAccessControl {

//********************************************************
//         Basic Types
//********************************************************

typedef sequence<boolean> BooleanList;

typedef Security::AttributeList AttributeList;


//********************************************************
//   Types that identify a secured resource
//********************************************************

typedef sequence<string> ResourceName;

typedef sequence<string> OperationList;

//*************************************************
//   Types associated with Administering Access Policy
//*************************************************
typedef string PolicyName;
typedef sequence<PolicyName> PolicyNameList;

const PolicyName NO_ACCESS_POLICY = "NO_ACCESS_POLICY";

interface DecisionCombinator;
interface PolicyEvaluator;
interface PolicyEvaluatorLocatorAdmin;
interface PolicyEvaluatorAdmin;

typedef sequence<PolicyEvaluator> PolicyEvaluatorList;

struct PolicyDecisionEvaluators {
   PolicyEvaluatorList    policy_evaluator_list;
   DecisionCombinator decision_combinator;
};

//*************************************************
//      Types used to request an Access Decision
//*************************************************

struct AccessDef {
   ResourceName   resource_name;
   string         operation;
};
typedef sequence<AccessDef> AccessDefList;

enum DecisionResult {ACCESS_DECISION_ALLOWED,
                     ACCESS_DECISION_NOT_ALLOWED,
                     ACCESS_DECISION_UNKNOWN
};
```

```
typedef sequence<DecisionResult> DecisionResultList;


//*************************************************
//     interface AccessDecision
//*************************************************

interface AccessDecision {

    boolean access_allowed(
        in ResourceName    resource_name,
        in  string         operation,
        in  AttributeList  attribute_list
    );

    BooleanList multiple_access_allowed(
        in  AccessDefList      access_def_list,
        in  AttributeList       attribute_list
    );

};

//*************************************************
//     interface DynamicAttributeService
//*************************************************

interface DynamicAttributeService {

    AttributeList get_dynamic_attributes(
        in  AttributeList   attribute_list,
        in  ResourceName   resource_name,
        in  string         operation
    );
};

//*************************************************
//     interface PolicyEvaluatorLocator
//*************************************************

interface PolicyEvaluatorLocator {

    readonly attribute PolicyEvaluatorLocatorAdmin
policy_evaluator_locator_admin;

    PolicyDecisionEvaluators get_policy_decision_evaluators(
        in  ResourceName    resource_name
    );

};

//*************************************************
//     interface PolicyEvaluatorLocatorAdmin
//*************************************************

interface PolicyEvaluatorLocatorAdmin {

    void add_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in  ResourceName resource_name
    );

    void replace_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in  ResourceName resource_name
    );

    void set_default_evaluators(
        in  PolicyEvaluatorList policy_evaluator_list
    );
```

```
    void apply_combinator (
        in  DecisionCombinator decision_combinator,
        in  ResourceName   resource_name
    );

    void set_default_combinator(
        in  DecisionCombinator decision_combinator
    );

};

//*****************************************************
//     interface PolicyEvalutator
//*****************************************************

interface PolicyEvaluator {

    readonly attribute PolicyEvaluatorAdmin policy_evaluator_admin;

    DecisionResult evaluate(
        in  ResourceName   resource_name,
        in  string         operation,
        in  AttributeList  attribute_list
    );

    DecisionResultList multiple_evaluate(
        in  AccessDefList  access_def_list,
        in  AttributeList  attribute_list
    );

};


//*****************************************************
//     interface PolicyEvalutatorAdmin
//*****************************************************

interface PolicyEvaluatorAdmin {

    void    replace_policy(
        in  PolicyName  policy_name,
        in  ResourceName resource_name
    );

    void    add_policy(
        in  PolicyName  policy_name,
        in  ResourceName resource_name
    );

    PolicyNameList list_policy();

    void    set_default_policy(
        in  PolicyName  policy_name
    );
};


//*****************************************************
//     interface DecisionCombinator
//*****************************************************

interface DecisionCombinator{

    boolean combine_decisions(
        in DecisionResultList decision_result_list
    );
};

};

#endif  // DfResourceAccessControl
```