OMG CORBAmed DTF

Healthcare Resource Access Control (HRAC)

Initial Submission

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

OMG TC Document corbamed/98-xx-xx
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.

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.

- 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

Access Decision Object (ADO) -

Secured Resource -

Client - Any system or application that accesses or requests service from a Person Identification Service.

Component - A cohesive set of software services

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*.

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 prototypes by at least one submitting organization prior to the final submission.

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 (Bob Blakley – current text by Carol Burt)

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(Konstantin)

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 (Konstantin Beznosov)

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 require and receive an authorization decision. The service is intended to be used by security-aware applications.

2.2 Problems Being Addressed

2.2.1 Obtaining Security Meta-data Associated With the Named Resource

Security meta-data is needed to make authorization decisions based on factors specific to the application domain. For example, in the case of patient medical records, a resource name passed to the HRAC service might be associated with a medical record (or its a part) of a particular patient. Person identifier of that patient could be directly or indirectly used by policy evaluators in the HRAC service. The submission team decided that security meta-data associated with the resource name is obtained elsewhere and is NOT passed along with the resource name. The proposed design policy evaluators and dynamic attribute service is made in such a way that some security meta-data can be obtained in the form of dynamic attributes, while other security meta-data can be obtained by policy evaluators. Please see section XX on the policy evaluators.

2.2.2 Syntaxes and Semantics of Resource Name

The proposed design defines syntaxes of resource name as a sequence of strings. The submission team decided that semantics of resource name should not be defined at all. Instead dynamic attributes are used to provide additional semantics associated with the named resource. Also, depending on the resource name different policy evaluators (and potentially different policies in those evaluators) can be used for making authorization decisions. Policy evaluators can carry knowledge about semantics associated with a given resource name.

2.2.3 Format of operations on resources

Operation name is used in making authorization decisions. The submitted design defines format of an operation as a string.

2.2.4 Integration with Existing Authorization Engines

Some times, a system vendor or an enterprise owner would like to leverage existing authorization mechanisms and integrate or re-use them with HRAC services. The problem of integrating existing access control engines is addressed in the submitted design by allowing use of arbitrary authorization engines as long as they provide required interface (PolicyEvaluator) for accessing them. An existing authorization engine or service can be "wrapped" to provide such an interface.

2.3 Problems Not Being Addressed

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

2.3.1 Understanding of Application Functionality or Data

Sometimes, intimate understanding of an application functionality is needed for proper exercising of access control policies. The submission team refrained from making HRAC service interfaces syntaxes or semantics to be 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 to be tailored to specifics of most applications' functionality and semantics of data they manipulate.

2.3.2 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 submission, there was no any candidate for an IDL interface that the submission team would agree upon. The specified model of HRAC service allows to add policy evaluators that implement different authorization policies. Those policy evaluators can have any interfaces and mechanism for expressing authorization rules governing access to secured resources.

2.3.3 Quality of Protection as Authorization Decision Factor

Sometimes, it is reasonable to grant particular access to secured resources only if 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 yet another factor in authorization decisions. The current version of the submission is not providing mechanisms that would allow to use 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 necessary mechanisms for this. The team might address this problem in the revised submission.

2.3.4 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.

2.4 Design Goals

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

2.4.1 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.

2.4.2 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.

2.4.3 SIMPLICITY:

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

2.4.4 GENERALITY:

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

2.4.5 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, and/or 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

2.4.6 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).

2.4.7 SCALABILITY:

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

2.5 Reference Model

Bubbles and interaction diagrams

(pretty picture goes here)

Reference Model for HRAC.

Nice works explaining model go here.

2.6 Discussion of Proposal Scope (Carol Burt)

To be written

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 PolicyLocator {
…
};

interface PolicyLocatorAdmin : PolicyLocator {
…
};

interface PolicyEvaluator {
…
};

interface PolicyEvaluatorAdmin : PolicyEvaluator {
…
};

interface DynamicAttributeService {
…
};

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 & Types used from Security
//*******************************************************

typedef sequence<boolean> Booleans;

enum Ternary {ACCESS_DECISION_ALLOWED, ACCESS_DECISION_NOT_ALLOWED,
ACCESS_DECISION_UNKNOWN};

typedef Security::AttributeList AttributeList;
```

Booleans

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.

Ternary

This type is used as a result in access decisions where access policy is applied. The Ternary meanings are:

ACCESS_DECISION_ALLOWED: access is ALLOWED based on the information provided.

ACCESS_DECISION_NOT_ALLOWED: access is NOT_ALLOWED based on the information provided

ACCESS_DECISION_UNKNOWN: an access decision cannot be determined based on the information provided.

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 identifies 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 NOPOLICY = "NOPOLICY";

interface PolicyEvaluator;

typedef sequence<PolicyEvaluator> PolicyEvaluatorList;
```

```
struct PolicyDecisionEvaluators {
    PolicyEvaluatorList    pevals;
    DecisionCombinator pcombinator;
};
```

PolicyName

A PolicyName is a string used to identify an AccessPolicy. This type is as an input parameter to the apply_policy(), and set_default_policy() methods of the PolicyEvaluatorAdmin interface.

NOPOLICY is a standard name used to indicate that a resource has no policy associated with access decisions. This is to distinguish from the case where a policy has not yet been applied.

PolicyNameList

A PolicyNameList is a sequence of PolicyName. This type 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 is a sequence of PolicyEvaluator object references and the DecisionCombinator returned from the get_policy_evaluators() method of the PolicyLocator interface. This contains all the references of objects that must be consulted during the 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> MultipleAccessDef;

struct DecisionResult {
    PolicyName     pname;
    Ternary        result;
};
typedef sequence<DecisionResult> DecisionResults;
```

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.

MultipleAccessDef

MultipleAccessDef 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

DecisionResult contains the PolicyName that was used to make a decision and the Ternary result of the decision. This is the type returned from the evaluate() method of the PolicyEvaluator.

DecisionResults

DecisionResults is a sequence of DecisionResult. This is the type returned from the multiple_evaluate() method of the PolicyEvaluator and 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.

```
TBD
```

3.2 AccessDecision interface

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

interface AccessDecision {

    boolean access_allowed(
        in ResourceName    rname,
        in  string         operation,
        in  AttributeList  attributes
    );

    Booleans multiple_access_allowed(
        in  MultipleAccessDef  requested_access,
        in  AttributeList       attributes
    );

};
```

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.

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 PolicyLocator interface

```
//*****************************************************************
//     interface PolicyLocator
//*****************************************************************

interface PolicyLocator {

    PolicyDecisionEvaluators get_policy_decision_evaluators(
        in  ResourceName    rname
    );

};
```

The PolicyLocator object is used to locate object the PolicyEvaluator objects 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.

get_policy_evaluators()

A PolicyDecisionEvaluators which contains a list of PolicyEvaluator object references and the combinator for this resource is returned to the client.

3.4 PolicyLocatorAdmin interface

```
//*****************************************************************
//     interface PolicyLocatorAdmin
//*****************************************************************

interface PolicyLocatorAdmin : PolicyLocator {
// use trading service to find evaluators.

    void apply_evaluator (
        in  PolicyEvaluator peval,
        in  ResourceName rname
    );

    void apply_combinator (
        in  DecisionCombinator pcombinator,
        in  ResourceName   rname
    );

    void set_default_evaluator(
        in  PolicyEvaluator peval,
        in  ResourceName   rname
    );

    void set_default_combinator(
        in  DecisionCombinator pcombinator
    );

};
```

The PolicyLocatorAdmin object is used to assoicate PolicyEvaluator objects with a ResourceName. This submission provides a framework for the support of one or more policy evaluators for a single resource.

apply_evaluator()

A PolicyEvaluator is added to the list of evaluators for the named resource. This evaluator will be in the list of PolicyEvaluators returned by the PolicyLocator get_policy_evaluators() method.

apply_combinator()

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

set_default_evaluator()

The PolicyEvaluator provided is set as a default for the named resource. This evaluator is now the only evaluator for the resource and will be the only evaluator returned by the PolicyLocator get_policy_evaluators() method. Additional PolicyEvaluators may be added with the apply_evaluator() method of this interface.

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 PolicyLocator get_policy_evaluators() method for these resources.

3.5 PolicyEvaluator interface

```
//*****************************************************************
//     interface PolicyEvaluator
//*****************************************************************

interface PolicyEvaluator {

    DecisionResult evaluate(
        in  ResourceName   rname,
        in  string         operation,
        in  AttributeList  attributes
    );

    DecisionResults multiple_evaluate(
        in  MultipleAccessDef  requested_access,
        in  AttributeList   attributes
    );

};
```

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

evaluate()

A single access decision is requested based on policy this evaluator determines is appropriate determining access to the named resource / operation given a list of 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 contains the name of the Policy that this evaluator used to make a decision and a Ternary result. The PolicyName is not required in the DecisionResult and may be a null string.

multiple_evaluate()

An access decision is made based on policy this evaluator determines is appropriate determining access to the named resource / operation pairs given the provided list of 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 sequence maps one to one in the same order to the provided sequence of ResourceName/operation pairs.

3.6 PolicyEvaluatorAdmin interface

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

interface PolicyEvaluatorAdmin : PolicyEvaluator{

    void    apply_policy(
        in  PolicyName  pname,
        in  ResourceName rname
    );

    PolicyNameList list_policy();

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

The PolicyEvaluatorAdmin interface is used to associate 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 a string. The PolicyEvaluatorAdmin interface allows those policies to be assigned "by name" to a secured resource. This interface is primarily provided for the application who wishes to assign a policy to a newly created resource programatically at the time of resource creation. It does, however, imply that the application have knowledge of the semantics of the policies in order to choose an appropriate policy for access to the secured resource.

apply_policy()

The policy identified by PolicyName is associated with the secured resource identified by the ResourceName.

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. An implementation must support the standard name of "NOPOLICY", but other policy names may be assigned as the default by the implementation.

3.7 DecisionCombinator interface

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

interface DecisionCombinator{

    boolean combine_decisions(
        in DecisionResults results
    );
};
```

The DecisionCombinator interface is used to combine the decisions of multiple PolicyEvaluators. Combinators may be provided with different behaviors. A combinator might support a simple "ANY" which means that if any of PolicyEvaluators returned ACCESS_DECISION_ALLOWED, the the return is TRUE. An alternative would be a DecisionCombinator that implemented "ALL" where all of the PolicyEvaluators must return ACCESS_DECISION_ALLOWED. DecisionCombinators could, alternatively, be very complex and take into account which policies returned ALLOWED or NOT_ALLOWED or UNKNOWN results in combining the decisions. For this reason, a combinator is a public interface and an appropriate implementation can be chosen or developed for each enterprise.

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 facility.

3.8 DynamicAttributeService interface

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

interface DynamicAttributeService {

    AttributeList get_dynamic_attributes(
        in  AttributeList attributes,
        in  ResourceName   rname,
        in  string         operation,
        in  PolicyEvaluatorList pevals
    );
};
```

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 the application to determine how the original AttributeList provided by the client should be modified.

`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. The returned AttributeList may be modified by this service. The service may add or remove SecAttributes to this list. It is this new list of SecAttributes that is used as the basis of access decisions by the HRAC.

4. DfHealthCareResource module (??)

This will be provided in revised submission

5. Conformance Classes (Bob Blakley)

6. Appendix - Use Case Examples

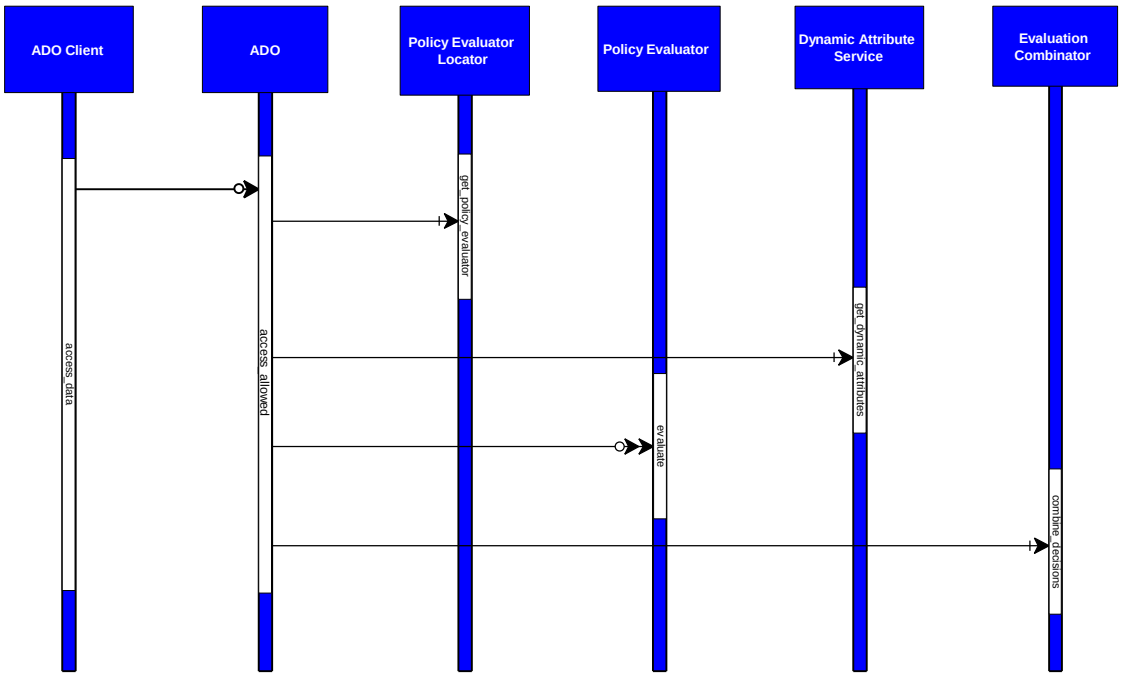
This appendix presents examples illustrating healthcare scenarios and the use of HRAC to provide access control for the instances of healthcare information access implied by these scenarios. Each example consists of several Use Cases:

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. A description of the actions of the healthcare application, the client of the Access Decision Object (ADO).
   B. A description of ADO actions.
   C. 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 dynamic 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.

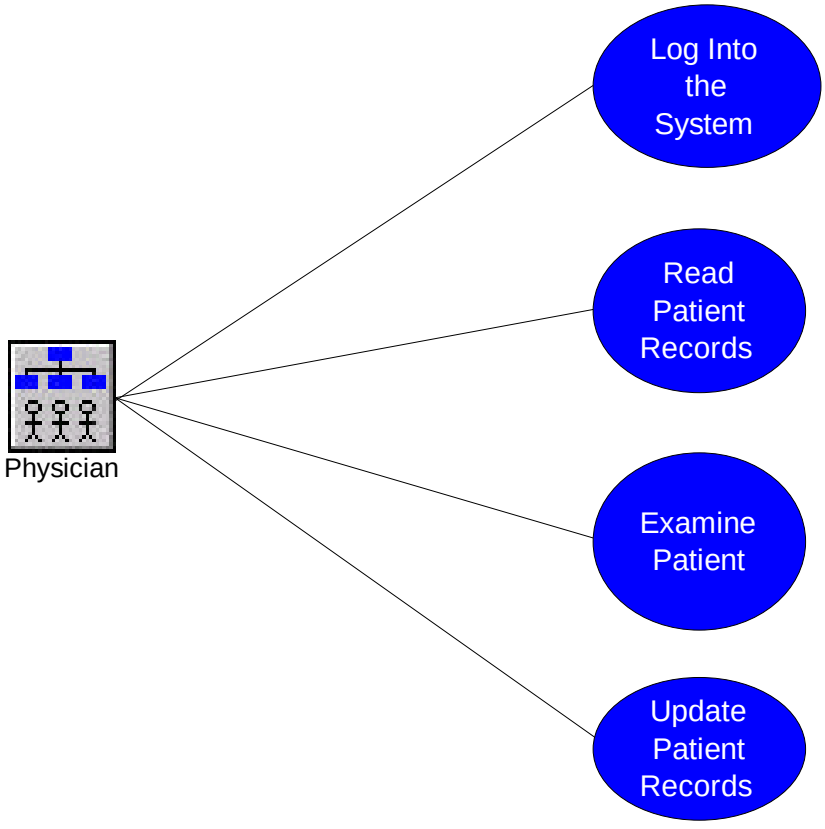
Depending on the implementation, a dynamic attribute may be the value of the dynamic attribute or a reference to the value of the dynamic attribute. If a reference, then the dynamic attribute value is obtained by the evaluator if and when the evaluator determines that the value is needed to make the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

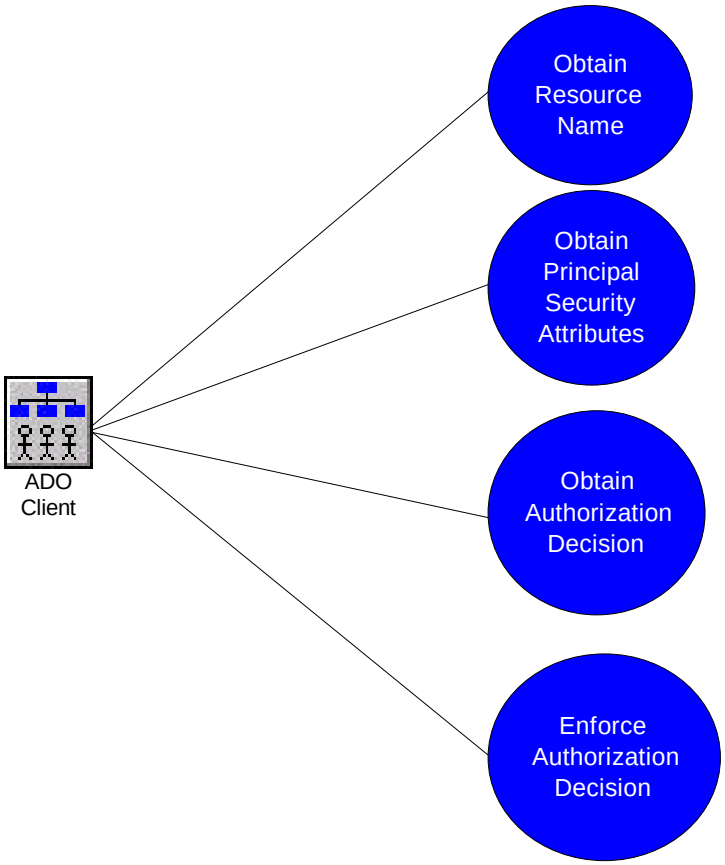
Table 1: Healthcare Scenario: Out-patient Visit to Attending Physician

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.

6.3 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.

6.4 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::PolicyLocator 2. Dynamic Attribute Service Object(DAS), which provides interface DfResourceAccessControl::DynamicAttributeService 3. Policy Evaluator Object (PE), which provides the interface DfResourceAccessControl::PolicyEvaluator 4. Policy 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

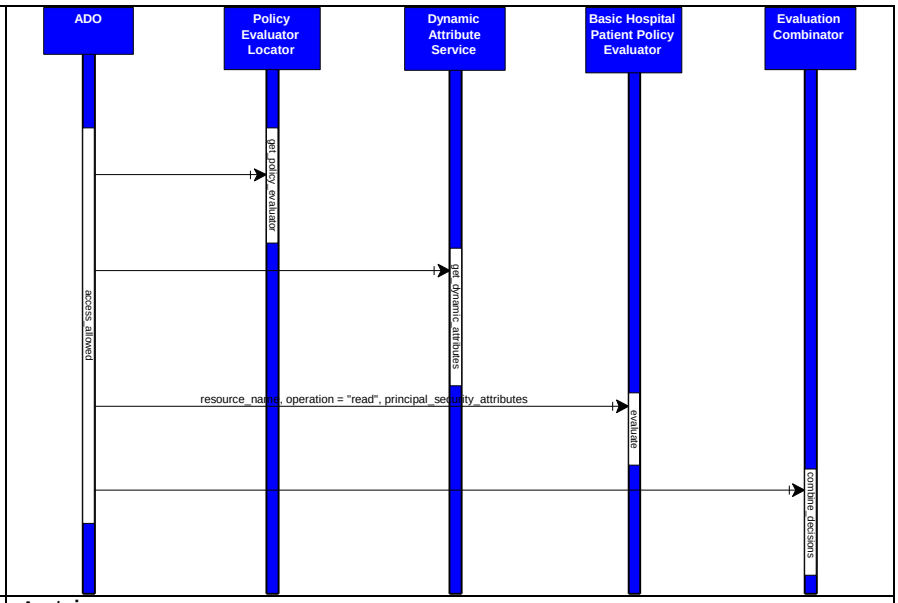
Diagram		
Description	Step	Action
	1	ADO invokes get_policy_evaluator(resource_name)which returns Policy_Evaluator for the Basic Hospital Patient Record Access Policy.
	2	ADO invokes get_dynamic_attributes(static_attributes, resource_name, operation) which returns attributes', a list of all static and 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({Policy_Evaluator, 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_evaluator() 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., YES, NO, UNKNOWN. If attributes' contains both the static attribute "physician" and the dynamic attribute "attending," then the result from policy_evaluator is YES 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.

6.5 ADO Object Interaction Diagram: Read Patient Record

This section shows the Object Interaction Diagram for ADO processing related to the Read Patient Record Use Case.

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 & Types used from Security
//********************************************************

typedef sequence<boolean> Booleans;

enum Ternary {ACCESS_DECISION_ALLOWED, ACCESS_DECISION_NOT_ALLOWED,
ACCESS_DECISION_UNKNOWN};

typedef Security::AttributeList AttributeList;


//***********************************************
//   Types that identifies 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 NOPOLICY = "NOPOLICY";

interface PolicyEvaluator;
typedef sequence<PolicyEvaluator> PolicyEvaluatorList;

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

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

struct DecisionResult {
   PolicyName     pname;
   Ternary        result;
};
typedef sequence<DecisionResult> DecisionResults;


//*********************************************************************
```

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

interface AccessDecision {

    boolean access_allowed(
        in ResourceName    rname,
        in  string         operation,
        in  AttributeList  attributes
    );

    Booleans multiple_access_allowed(
        in  MultipleAccessDef  requested_access,
        in  AttributeList       attributes
    );

};

//*****************************************************************
//     interface PolicyLocator
//*****************************************************************

interface PolicyLocator {

    PolicyEvaluatorList get_policy_evaluators(
        in  ResourceName    rname
    );

};

//*****************************************************************
//     interface PolicyLocatorAdmin : PolicyLocator
//*****************************************************************

interface PolicyLocatorAdmin  {
// use trading service to find evaluators.

    void apply_evaluator (
        in  PolicyEvaluator peval,
        in  ResourceName rname);

    void set_default_evaluator(
        in  PolicyEvaluator peval);

};

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

interface PolicyEvaluator {

    DecisionResult evaluate(
        in  ResourceName   rname,
        in  string         operation,
        in  AttributeList  attributes
    );

    DecisionResults multiple_evaluate(
        in  MultipleAccessDef  requested_access,
        in  AttributeList   attributes
    );

};

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

interface PolicyEvaluatorAdmin {
```

```
// can apply NOPOLICY

    void    apply_policy(
        in  PolicyName  pname,
        in  ResourceName rname
    );

    PolicyNameList list_policy();

    void    set_default_policy(
        in  PolicyName  pname
    );
};

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

interface DecisionCombinator{

    boolean combine_decisions(
        in DecisionResults results
    );
};

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

interface DynamicAttributeService {

    AttributeList get_dynamic_attributes(
        in  AttributeList attributes,
        in  ResourceName   rname,
        in  string         operation,
        in  PolicyEvaluator peval
    );
};



};

#endif  // DfResourceAccessControl
```