OMG CORBAmed DTF

Healthcare Resource Access Control (HRAC)

Initial Submission

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

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.

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 RFP1, Patient Identification Service (PIDS), Object Management Group (OMG) document number corbamed/96-11-08.

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 Miami, FL
V. "Juggy" Jagannathan CareFlow|Net, Inc. 235 High Street, Suite 225 Morgantown, WV 26505 304 293 7535 juggy@careflow.com	Bob Blakley IBM Austin, TX

1.2 Supporting Organizations (Dave Chizmadia)

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 Security Agency
- Philips Medical Systems
- ???

1.3 Conventions

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

1.4 Terminology (Dave Chizmadia)

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 (Bob Blakley – current text by Carol Burt)

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 (Carol Burt)

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)

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

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.

DISCUSSION POINTS REQUESTED

How new CORBAMed specifications will employ the submitted specification .

> To be written

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

> To be written

Mechanisms provided for extensibility

> To be written

2. Overview of Response

2.1 Introduction (Konstantin Beznosov)

To be written

2.2 Problems Being Addressed(Konstantin Beznosov)

To be written.

2.3 Problems Not Being Addressed(Konstantin Beznosov)

To be written

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

2.4 Design Goals (Bob Blakley)

To be written (see e-mail list for starter goals)

2.5 Reference Model (Dave Chizmadia)

(pretty picture goes here)

Reference Model for HRAC.

Nice works explaining model go here.

2.6 Discussion of RFP Scope (Carol Burt)

To be written

2.7 HRAC Frequently Asked Questions (FAQ)

From e-mail list

2.8 General Usage Discussion (Bob Blakley)

To be written

2.9 Healthcare Specific Usage Scenarios (Konstantin Beznosov, John Barkley, Juggy Jaganathan)

To be written

2.10 HRAC Conceptual Model (Carol Burt)

To be written

3. DfResourceAccessControl module (C Burt)

```
//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 DecisionRulesAccess {
…
};

interface DecisionRulesAdmin : DecisionRulesAccess {
…
};

interface DynamicEvaluator {
…
};

};

#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
Security::Right
Security::RightsList
Security::RightsCombinator
```

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;

typedef TimeBase::IntervalT IntervalT;
typedef sequence<IntervalT> TimeIntervals;

typedef Security::AttributeList AttributeList;

const unsigned short HRAC_RIGHTS_FAMILY_DEFINER = 256;
const unsigned short HRAC_RIGHTS_FAMILY_STATIC = 0;
const unsigned short HRAC_RIGHTS_FAMILY_DYNAMIC = 1;

typedef Security::Right Right;
typedef Security::RightsList RightsList;
typedef Security::RightsCombinator RightsCombinator;
```

Booleans

A sequence of boolean used as a return value when multiple decisions are requested. This type is used as a return value in AccessDecision and DynamicEvaluator interface methods.

IntervalT

TBW

TimeIntervals

TBW

AttributeList

The Security::AttributeList is defined as follows in CORBA Security 1.2 (ptc/98-01-02).

```
typedef sequence<octet> Opaque;

// security attributes
typedef unsigned long SecurityAttributeType;

// other attributes; family = 0
const SecurityAttributeType    AuditId = 1;
const SecurityAttributeType    AccountingId = 2;
const SecurityAttributeType    NonRepudiationId = 3;

// privilege attributes; family = 1
const SecurityAttributeType    Public = 1;
const SecurityAttributeType    AccessId = 2;
const SecurityAttributeType    PrimaryGroupId = 3;
const SecurityAttributeType    GroupId = 4;
const SecurityAttributeType    Role = 5;
const SecurityAttributeType    AttributeSet = 6;
const SecurityAttributeType    Clearance = 7;
```

```
const SecurityAttributeType    Capability = 8;

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;
```

HRAC_RIGHTS_FAMILY_DEFINER

This constant is a standard family identifier for Rights that may be used by implementations of the HRAC. Usage of this definer in the ExtensibleFamily struct of a Right identifies the Right as unique to the HRAC domain.

HRAC_RIGHTS_FAMILY_STATIC

This constant is a standardized family within the HRAC definer shown above for Rights that are statically linked to the SecAttribute. Other families may be defined by an HRAC implementation and treated as static, but this family is provided for interoperable rules. This family should only be used with the HRAC_RIGHTS_FAMILY_DEFINER.

HRAC_RIGHTS_FAMILY_DYNAMIC

This constant is a standardized family within the HRAC definer shown above for Rights that are dynamic and can only be determined at access decision time. Other families may be defined by an HRAC implementation and treated as dynamic, but use of this family is provided for interoperable rules. This family should only be used with the HRAC_RIGHTS_FAMILY_DEFINER.

Right

An explanation of Rights and Rights Families is in section 15.6.4 on pages 15-123 and 15-124 of the CORBA Security 1.2 specification (ptc/98-01-02). The SecurityAdmin::AccessPolicy interface is described in section 15.6.4.3 on page 15-126 of that specification. A detailed description of how access decisions are made using a required rights model in section 15.6.4.5 starting on page 15-127.

The Security::Right is defined as follows in CORBA Security 1.2 (ptc/98-01-02).

```
struct Right {
      ExtensibleFamily        rights_family;
      string                  right;
};
```

An implementation may use existing RightsFamilies and implement a direct coorelation to the CORBASec required rights model, or may choose to use a RightsFamily defined by this specification to provide a static mapping from a SecAttribute to a right used for resource access control. A RightsFamily is defined above for HRAC static and dynamic rights.

RightsList

```
typedef sequence <Right> RightsList;
```

RightsCombinator

```
enum RightsCombinator {
        SecAllRights,
        SecAnyRight
};
```

A RightsCombinator when used in combination with a RightsList specifies how the elements of the list are related for the purpose of access decisions. An OperationRuleComponent in this specification uses a combinator and rights list to specify one component of a decision rule on an operation of a secured resource.

3.1.2 Types that identify and manage information about secured resources

```
//*********************************************************************
//   Type that identifies a Secured Resource
//*********************************************************************

typedef sequence<string> ResourceName;
```

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. Note: We should discuss whether we want to make this a stronger requirement by putting the type in the IDL.

3.1.3 Types related to Resource Access Decision Rules

```
//*************************************************
//   Types related to Resource Access Decision Rules
//*************************************************

struct OperationRuleComponent {
       RightsCombinator   combinator;
       RightsList         rights;
};
typedef sequence<OperationRuleComponent> OperationRule;

enum   PermissionControlModel { GRANT, DENY };

struct Rule {
       PermissionControlModel control_model;
       OperationRule          the_rules;
};

struct ResourceRuleComponent {
       string             operation;
       OperationRule      the_rule;
```

```
};
typedef sequence<ResourceRuleComponent> ResourceRules;

struct Rules {
       PermissionControlModel control_model;
       ResourceRules          the_rules;
};
```

NOTE: For all the examples below, the RightsList includes only the string part of a Right. Assume for simplicity that all of the RightsFamily are 256,0 (HRAC_RIGHTS_FAMILY_DEFINER : HRAC_RIGHTS_FAMILY_STATIC). A discussion of dynamic rights will be provided in the section on the DynamicEvaluator. The tables below are intended to be informative and are not intended to imply a standard mapping from a SecAttribute to a Rights string. Other examples of rules based on a traditional CORBASec RightsList are possible and not precluded by this specification.

`OperationRuleComponent`

An operation rule component specifies a single component of an operation **access decision rule.** For example, the component may be as follows:

Example RuleComponent:

RightsCombinator	SecAllRight
RightsList (256,0)	Accessid:carol, role:architect, role:executive

`OperationRule`

An operation rule is a sequence of OperationRuleComponent. There is an implied "OR" between the members of the sequence. For example, if a rule had a sequence of two OperationRuleComponents as follows:

Example OperationRule:

RuleComponent	**RightsCombinator**	SecAllRight
	RightsList (256,0)	Accessid:carol, role:architect, role:executive
RuleComponent	**RightsCombinator**	SecAnyRight
	RightsList (256,0)	Accessid:bob, role:goodguy

The OperationRule would be interpreted as

((accessid:carol & role:architect & role:executive) | accessid:bob | role:goodguy)

ISSUE: It must be determined what happens if a rule is inconsistent. For example:

((accessid:carol & role:architect & role:exectutive) | accessid:carol | role:goodguy)

I propose that any Right that appears in a RightsList where the combinator is "SecAnyRight" cannot exist in a RightsList within the OperationRule sequence where the combinator is

"SecAllRight". This would force the inconsistent rule above to be expressed concisely as ((role:architect & role:executive) | accessid:carol | accessid:bob | role:goodguy)

`PermissionControlModel`

Resource Decision Rules have a grant/deny PermissionControlModel that is expressed as part of the definition of the rules. GRANT means: the decision rules for this resource are used to GRANT access. DENY means: the decision rules for this resource are used to DENY access. So, the control is expressed on the resource decision rule and tells the service how to evaluate the rule against effective rights. The way that a decision is made is that the effective_rights of the principle is compared against the rule. If the PermissionControlModel is GRANT, then the access is granted (access_allowed = TRUE) if the rights evaluated against the rules are true. If the PermissionControlMode is DENY, then access is denied (access_allowed = FALSE) if the rights evaluated against the rules are true.

The "decision rules" are specified by defining the rights required to access a secured resource without regard to how rights are assigned to a SecAttribute.

`Rule`

A Rule contains the PermissionControlModel for the operation and the OperationRule for a single operation. This is the type returned from the get_rule method where the client is asking for a single rule for a Resource/Operation pair. The PermissionControlModel is at the Resource level, but is repeated in a Rule so that an Access Decision Object could retrieve if desired the rule for a single operation of a resource without retrieving all rules for the resource or making a separate method invocation to get the permission control model.

Example Rule:

PermissionControl Model	GRANT	
RuleComponent	**RightsCombinator**	SecAllRight
	RightsList (256,0)	Accessid:carol, role:architect, role:executive
RuleComponent	**RightsCombinator**	SecAnyRight
	RightsList (256,0)	Accessid:bob, role:goodguy

`ResourceRuleComponent`

A ResourceRuleComponent specifies a single component of the access decision rules for a secured resource. It includes the operation and the OperationRule. This type is only used to build other types and is not used directly in any IDL interface.

`ResourceRules`

ResourceRule is a sequence of all the ResourceRule components for a secured resource

`Rules`

Rules contain the PermissionControlModel for the resource and the ResourceRules.

Example Rules:

PermissionControl Model	GRANT	
Operation	CREATE	
RuleComponent	**RightsCombinator**	SecAnyRight
	RightsList (256,0)	Accessid:bob, role:goodguy
Operation	ACCESS	
RuleComponent	**RightsCombinator**	SecAllRight
	RightsList (256,0)	role:architect, role:executive
RuleComponent	**RightsCombinator**	SecAnyRight
	RightsList (256,0)	Accessid:bob, role:goodguy
Operation	UPDATE	
RuleComponent	**RightsCombinator**	SecAllRight
	RightsList (256,0)	Accessid:carol, role:architect, role:executive
RuleComponent	**RightsCombinator**	SecAnyRight
	RightsList	Accessid:bob, role:goodguy
Operation	DELETE	
RuleComponent	**RightsCombinator**	SecAllRight
	RightsList (256,0)	Accessid:carol, role:architect, role:executive
RuleComponent	**RightsCombinator**	SecAnyRight
	RightsList (256,0)	Accessid:bob, role:goodguy

3.1.4 Types used to request an Access Decision

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

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

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. See the multiple_access_allowed() method of the AccessDecision interface.

3.1.5 Types related to Dynamic Evaluation of Rights

```
//*************************************************
//   Types related to Dynamic Evaluation of rights
//*************************************************

typedef sequence<octet> ResourceKey;

struct DynamicInfo {
    DynamicEvaluator       evaluator;
    ResourceKey            key;
    RightsList             dyn_rights;
};
```

ResourceKey

The ResourceKey is registered as a resource identifier when a dynamic evaluator is used. See the usage in the DecisionRulesAdmin set_dynrights_support() and the DynamicEvaluator evauate() methods. This is the only usage of this type – it is merely saved by HRAC and provided as a parameter with the invocation of a method on a dynamic evaluator. This is a convenience to the application who may have an internal identifier of the secured resource that is more convienient to use than a ResourceName at the time the dynamic evaluator is called.

DynamicInfo

This structure contains all of the information necessary to register a dynamic evaluator. It contains object reference of the DynamicEvaluator, the ResourceKey of the resource, and a list of dynamic rights which it can be evaluated by the evaluator for the resource. See the set_dynrights_support() method of the DecisionRulesAdmin interface.

3.1.6 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. Since decision rules are expressed using Rights, it is necessary for the Access Decision object to determine as part of it's implementation the effective rights associated with the AttributeList. It is, however, out of the scope of this submission to mandate how the ADO determines the effective rights of the principal based on the SecAttributes.

`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 DecisionRulesAccess interface

```
//*************************************************
//     interface DecisionRulesAccess
//*************************************************

interface DecisionRulesAccess {

    DynamicInfo get_dynrights_support(
        in ResourceName rname
    );

    Rule   get_effective_rule(
        in ResourceName rname,
        in string      operation
    );

    Rules  get_effective_rules(
        in ResourceName rname
    );

};
```

The DecisionRulesAccess allows an Access Decision Object to retrieve the decision rules and other information necessary to make an access decisions.

get_dynrights_support()

This provides to the client an object reference of the DynamicEvaluator, a ResourceKey, and a list of the dynamic rights that can be evaluated for this resource.

get_effective_rule()

The effective Rule for a single operation on a secured resource is provided with this method. The word "effective" is meant to imply that there may be elements of the rule that are defined but not part of the rule effective at the time of invocation. This might be true if some components of the rule are time-based. For example, a component of a rule may only be valid from 8 am to 5 pm on Monday through Friday.

get_effective_rules()

All of the rules for a secured resource are provided with this method.

ISSUE: Should this operation return an iterator?

3.4 DecisionRulesAdmin interface

```
//*******************************************************
//     interface DecisionRulesAdmin : DecisionRulesAccess
//*******************************************************
interface DecisionRulesAdmin : DecisionRulesAccess {

    void set_resource_key (
        in  ResourceName rname,
        in  ResourceKey key
    );

    void set_dynrights_support (
        in  ResourceName rname,
        in  DynamicInfo dynamic_info
    );

    void set_rule (
        in PermissionControlModel control_model,
        in ResourceName    rname,
        in string          operation,
        in TimeIntervals   effective_time,
        in Rule            the_rule
    );

};
```

The DecisionRulesAdmin interface extends the DecisionRulesAccess interface …

set_resource_key()

This associates a ResourceKey with a ResourceName. This key will be used in parameters of the DynamicEvaluator instead of the ResourceName.

set_dynrights_support()

This sets the information necessary for dynamic evaluation of rights.

set_rule()

This allows the definition of decision rules.

ISSUE: Time needs to be expanded for the rule definition.

3.5 DynamicEvaluator interface

```
//*****************************************************************
//     interface DynamicEvaluator
//*****************************************************************

interface DynamicEvaluator {

    boolean evaluate(
        in  ResourceKey     resource_key,
        in  RightsList     effective_rights,
        in  Right          dynamic_right);

    Booleans multiple_evaluate(
        in  ResourceKey     resource_key,
        in  RightsList     effective_rights,
        in  RightsList     dynamic_right);
};
```

The DynamicEvaluator interface. Note: Need an example.

evaulate()

The evaulator determines if, the dynamic_right is currently in force based on the resource identified and the effective rights passed to the evaluator

Multiple_evaluate()

This is a request for multiple evaluations with a single method call.

4. DfHealthCareResource module (??)

This should contain the healthcare specific ResourceNames that are standardized in this specification.

5. Conformance Classes (Bob Blakley)

6. Appendix - Use Case Examples (Konstantin Beznosov, John Barkley, Juggy Jaganathan)

7. Appendix - Complete IDL (Carol Burt)

```
//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 {

//*********************************************************
//*                  Forward interfaces                   *
//*********************************************************
interface DynamicEvaluator;

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

typedef sequence<boolean> Booleans;

typedef TimeBase::IntervalT IntervalT;
typedef sequence<IntervalT> TimeIntervals;

typedef Security::AttributeList AttributeList;
typedef Security::Right Right;
typedef Security::RightsList RightsList;
typedef Security::RightsCombinator RightsCombinator;


//*************************************************************************
//   Types that identify and manage information about secured resources
//*************************************************************************

typedef sequence<string> ResourceName;


//*****************************************************
//   Types related to Resource Access Decision Rules
//*****************************************************

struct OperationRuleComponent {
       RightsCombinator   combinator;
       RightsList         rights;
};
typedef sequence<OperationRuleComponent> OperationRule;

enum   PermissionControlModel { GRANT, DENY };

struct Rule {
       PermissionControlModel control_model;
       OperationRule          the_rules;
};

struct ResourceRuleComponent {
       string             operation;
       OperationRule      the_rule;
};
typedef sequence<ResourceRuleComponent> ResourceRules;

struct Rules {
```

```
        PermissionControlModel control_model;
        ResourceRules          the_rules;
};

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

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

//*****************************************************
//   Types related to Dynamic Evaluation of rights
//*****************************************************

typedef sequence<octet> ResourceKey;

struct DynamicInfo {
    DynamicEvaluator       evaluator;
    ResourceKey            key;
    RightsList             dyn_rights;
};



//*********************************************************************
//     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 DecisionRulesAccess
//*************************************************

interface DecisionRulesAccess {

    DynamicInfo get_dynrights_support();

    Rule   get_effective_rule(
        in ResourceName rname,
        in string      operation
    );

    Rules  get_effective_rules(
        in ResourceName rname
    );

};

//*****************************************************
//     interface DecisionRulesAdmin : DecisionRulesAccess
```

```
//*********************************************************
interface DecisionRulesAdmin : DecisionRulesAccess {

    void set_resource_key (
        in  ResourceName rname,
        in  ResourceKey key
    );

    void set_dynrights_support (
        in  DynamicInfo dynamic_info
    );

    void set_rule (
        in PermissionControlModel control_model,
        in ResourceName    rname,
        in string          operation,
        in TimeIntervals   effective_time,
        in Rule            the_rule
    );

};



//*****************************************************************
//     interface DynamicEval
//*****************************************************************

interface DynamicEvaluator {

    boolean evaluate(
        in  ResourceKey     resource_key,
        in  RightsList     effective_rights,
        in  Right          dynamic_right);

    Booleans multiple_evaluate(
        in  ResourceKey     resource_key,
        in  RightsList     effective_rights,
        in  RightsList     dynamic_right);
};


};
#endif  // DfResourceAccessControl
```