Carol Burt
2AB, Inc.
09/28/98

Hi,

Round 2.

The IDL explained below is what I believe is a minimal set of interfaces to meet the requirements I've heard from a lot of the medical people.

The things that I tried to do based on feedback in Seattle:

1) Limit the administrative interfaces to those specifically needed to meet the requirements of the RFP – this will allow more flexibility in implemetation choices based on security policys and support for multiple rules languages in vendor products.

   I have partitioned the administration issues into distinct areas and then declared some of those areas "out of scope" The partitioning is as follows:

   a) Administration of Access Policy - defined as "the granting of Rights to SecAttributes" (see DomainAccessPolicy in CORBASec).
   b) Administration of Resource Definitions – defined as the assignment of Names and operations to secured resources and the grouping of those resources into a hierarchy if appropriate.
   c) Administration of Decision Rules for secured resources – defined as creating a set of rules that are used at access decision time to determine whether access should be allowed. This also includes the ability to provide an external evaluator that may be consulted at access decision time.

   I am suggesting that we declare items a and b "out of scope". Item c is "in scope".

   Here is my "out of scope" reasoning:

   a) Administration of Access Policy – an Access Policy could be used by the ADO at access decision time (get_effective_rights) or a implementation might simply support a static mapping from a SecAttribute to a Right that is a stringified version of the SecAttribute. We could standardize how this mapping is done to ensure interoperability if people think it is a good idea. As I see it, this would allow flexibility for very simple access decision rules based only on the ResourceName/Operation and SecAttributes within the existing rights model of CORBASec. It would not preclude, however, the required rights model where many "Rights" are granted to a SecAttribute.

   To declare this "out of scope" we have to agree that the "control" HRAC exerts is only the resource. 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. For example, if we want to support GRANT/DENY policy (which means: the decision rules for this resource are used to GRANT access, or the decision rules for this resource are used to DENY access), the control is expressed on the resource decision rule and tells the service how to evaluate the rule against effective rights.

b) Administration of Resource Definitions – it is very likely that an implementation will provide a tool to define resources and their associated operations as pre-requisite to defining decision access rules for Resource/operations. We could argue, however, that just as the administration of Users (defining SecAttributes) and the assignment of Rights to SecAttributes is "out of scope", the interfaces to administering a repository of resource information (including hierarchy's of resources) is "out of scope". An implementation is expected to use such a repository, just as it will use a repository of User/SecAttribute information, but it could be "out of scope" for the HRAC service. What must be "in scope" (imo) are methods necessary for a decision rule to be defined dynamically so that applications can dynamically add decision rule information. Without a standard way to do this, the service would be of very limited use to applications which is why I say c) is "in scope".

I have done some work on interfaces for the "out of scope" areas, but the IDL is not included here. I am including at the end a visual for those who find it useful to see what an implementation might think about supporting. This is NOT intended to be part of the submission.

2) Make a clean separation of Access interfaces and Admin interfaces and relate them via inheritance so that conformance points can be easily defined.

   This has been done for the DecisionRulesAccess and DecisionRulesAdmin:DecisionRulesAccess interfaces defined below.

   I also factored other interfaces in my initial draft defining a RepositoryAccess and a RepositoryAdmin and a AccessPolicy and AccessPolicyAdmin but they are NOT included here because of the "out of scope" discussion above.

3) Add more flexibility in the dyanamic evaluation interface to accommodate evaluations other than those based on dynamic security attributes

   This interface has been changed to request an evaluation based on dynamic rights, not SecAttributes. Since a Right is extensible and qualified by families this should allow a great deal of flexibility. For example, this would allow an application to define a family of rights as an expression if they wished and evaluate that expression at runtime. The HRAC would not need to have any understanding of the format or evaluation rules for dynamic rights.

4) Allow both a DENY and GRANT access policy for resources (There is NOT a consensus for including this, but strong input from users caused me to include it in this draft).

   The 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.

For example, consider where the attribute list evaluates to the following effective rights(however the ADO determines them):

Access_id:carol, role:architect, role:2ab-executive

If the decision rule for the resource/operation requested is:

```
----- Decision Rule ----
   Any:  access_id:bob, role:goodguy
   All:  access_id:carol, role:architect
------------------------
```

If PermissionControlModel = GRANT then access_allowed = TRUE
If PermissionControlModel = DENY then access_allowed = FALSE

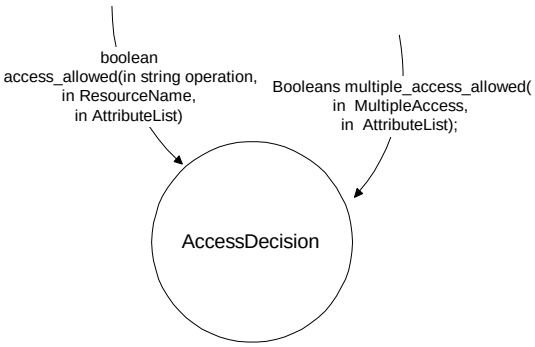
5) Accommodate sophisticated decision rules expressed in a structured way (not a rules language).

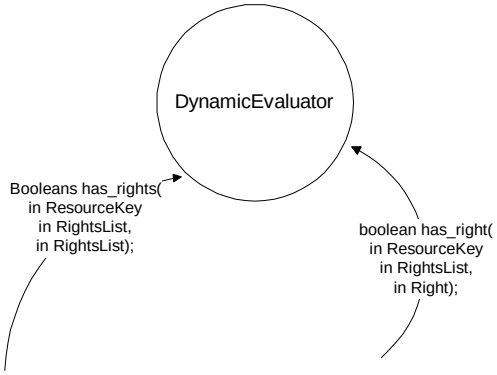
Rules are structured and contain sequences of RightsLists each of which have combinators. See example above. I’d like feedback on what type of decision rules people believe cannot be expressed this way.

6) Accommodate rules that are defined by a direct mapping from SecAttribute or by using the granted_rights model in CORBASec.

If you defined a direct mapping from a SecAttribute to a stringified version that was a “granted right”, you’ve effectively flattened this to a direct mapping without changing the access policy model in CORBASec. See explanation above of why I considered administration of Access Policy “out of scope”.

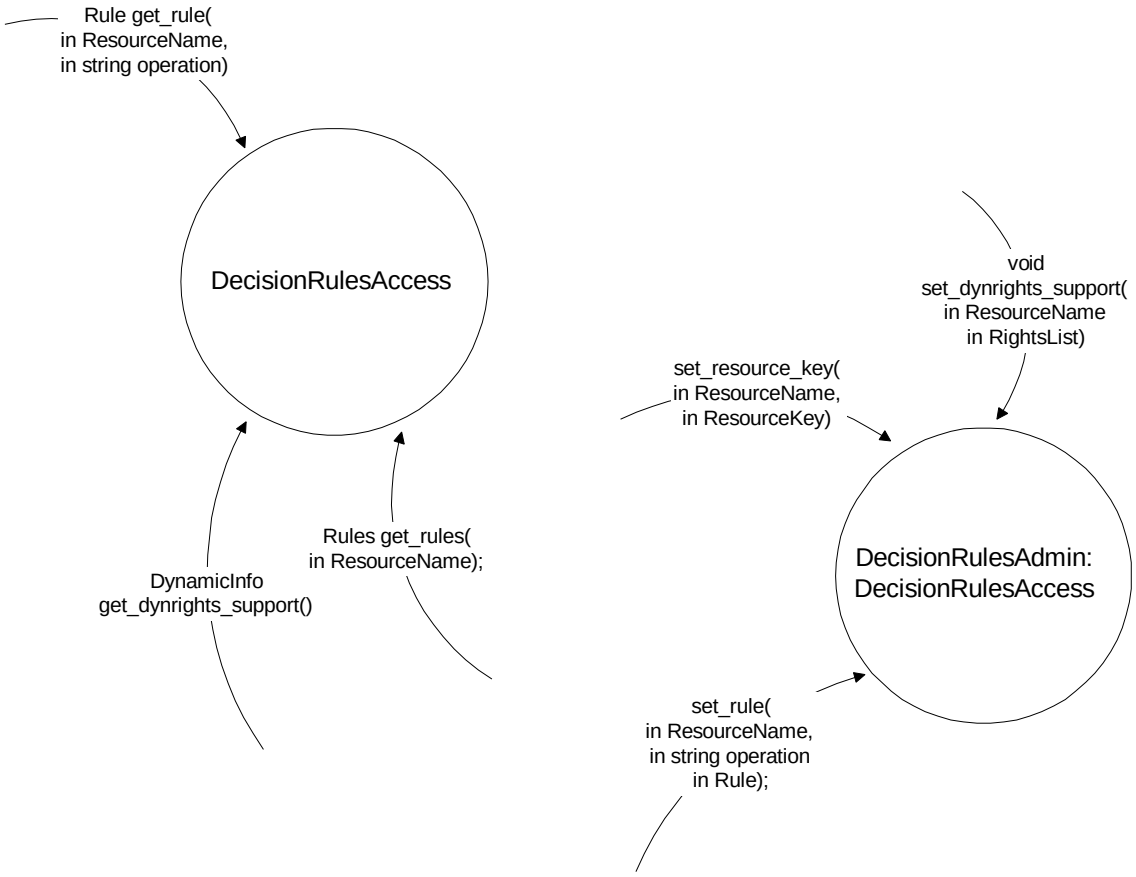
A visual of the interfaces follows, then the IDL.


boolean
access_allowed(in string operation,
in ResourceName,
in AttributeList)
Booleans multiple_access_allowed(
in MultipleAccess,
in AttributeList);
AccessDecision


DynamicEvaluator
Booleans has_rights(
in ResourceKey
in RightsList,
in RightsList);
boolean has_right(
in ResourceKey
in RightsList,
in Right);

Carol Burt
2AB, Inc.
09/28/98

Decision Rules Access & Administration



Carol Burt
2AB, Inc.
09/28/98

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

#ifndef _DF_RESOURCE_ACCESS_CONTROL_IDL_
#define _DF_RESOURCE_ACCESS_CONTROL_IDL_

#include <orb.idl>

#include "Security.idl"
#include "SecurityLevel2.idl"
//#include "NamingAuthority.idl"

This proposal does not use the NamingAuthority at the moment(this is
used in all CORBAMed proposals, however, it is required for HRAC
implementations that the first string in the ResourceName be structured
as a NamingAuthority::QualifiedNameStr for interoperability with other
CORBAMed specifications that use this module.

#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;

//*********************************************************
//   extensible families for standard data types
//**
//  struct ExtensibleFamily {
//      unsigned short       family_definer;
//      unsigned short       family;
//  };
//**
//   Declarations related to Rights
//**
//  struct Right {
//      ExtensibleFamily       rights_family;
//      string                 right;
```

```
// };
//
// typedef sequence <Right> RightsList;
//
// enum RightsCombinator {
//       SecAllRights,
//       SecAnyRight
// };
//*************************************************************

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

typedef sequence<string> ResourceName;

typedef sequence<ResourceName> ResourceNameList;

ResourceName is a sequence of string.  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. We should discuss whether we want to make this a
stronger requirement by putting the type in the  IDL.

typedef sequence<string> OperationList;

I went back to the term “operation” for what was called intent in my
earlier draft.  Operations are on secured resources and cover actions
like create, update, delete, use… or whatever clever things one might
want to do to a resource.

//*****************************************************
//   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;
};

Combinators are applied to the rights list.  A sequence of these rules
components make up a single rule.  There is an implied “OR” implied
between the elements of the sequence.   The PermissionControl is at the
Resource level but is repeated here so a single rule is concise when
ADO retrieves it singularly.
```

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

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

Rules contain all the rules for a resource. Rules are retrieved as "effective rules" so time does not have to be an element in the rule despite the fact that the administrative interfaces may specify time based decision rules. The PermissionControlModel is only specified once when all rules are retrieved.

```
//*********************************************************
//    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;
```

registered as a resource identifier when a dynamic evaluator is used. This is the only usage of this type – it is only saved by HRAC to be provided with the invocation of a dynamic evaluator.

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

Dynamic Evaluators may be provided that can evaluate the dynamic rights listed. Since Right contains a string, it is up to the evaluator to parse and understand the meaning of the "right".

Carol Burt
2AB, Inc.
09/28/98

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

This is the AccessDecision interface.  As agreed, this takes an
AttributeList instead of Credentials and is not locality constrained.

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
//****************************************************

The minimal interface to be able to retrieve rules necessary to make
access decisions are this one and (perhaps depending on implementation
a DomainAccessPolicy to retrieve "effective_rights" – see note at
front);

interface DecisionRulesAccess {

        DynamicInfo get_dynrights_support();

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

        Rules   get_effective_rules(
                in ResourceName rname
        );

};
```

Carol Burt
2AB, Inc.
09/28/98

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

The minimal administrative interface to allow dynamic creation of decision rules and to set up dynamic evaluators. The time expressed below must be modified to meet the requirements of healthcare decision rules.

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

It is expected (but other options exist) that this interface will typically be implemented by the application itself or by the enterprise in which the application is installed. The idea is that the application (or some relationship service that is used by the application, or some local logic) is the only place that truly understands the relationships that are necessary to determine if a Dynamic Right is currently valid. For example, we may not want or be able to "configure" the rights associated with an "ATTENDING_PHYSICIAN" SecAttribute. The application at runtime knows how to determine if this is true; the HRAC doesn't. The application may use a relationship service, or it may "look" in the patient record, or it may know it is true some other way. So the idea is that for a Resource, a list of Dynamic Rights that are part of the decision logic will be configured (we may wish to define a rights family for this). These will include things like "ATTENDING_PHYSICIAN". The application (or some other service -- internal or external to HRAC depending on implementation) will use the set_dynrights_support () operation of the DecisionRulesAdmin to give the HRAC an object reference of an evaluator and a ResourceKey associated with a particular ResourceName that can be used to ask if a [particular set of] right(s) is currently true. Note that this can be a relationship based right but it isn't restricted to

that. It could be any Right that can be determined only by the application or by logic written for a particular installation – and it could include an expression understood by the evaluator.

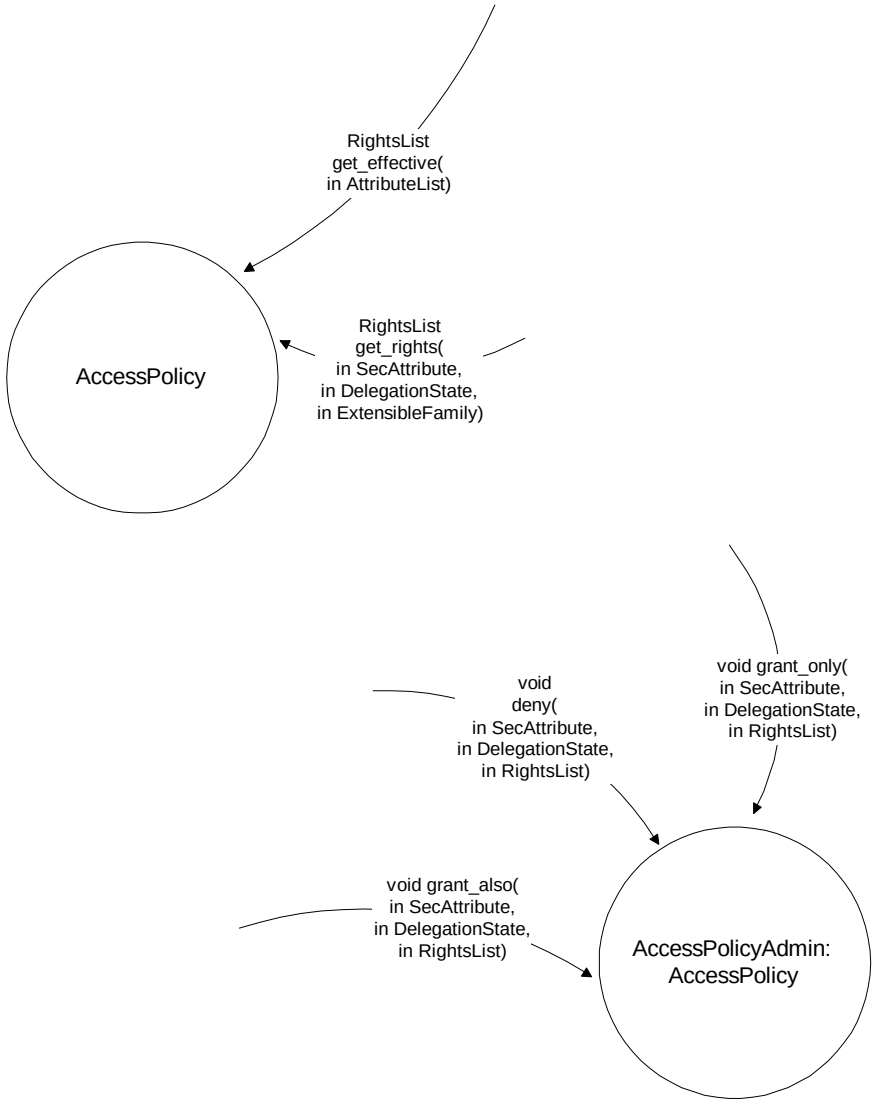
The rights are expressed in the rules in the same way as any Right .. we just have to find out if the dynamic right is currently true.

```
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
```

Carol Burt
2AB, Inc.
09/28/98

The following are the “out of scope” objects. These are not expected to be part of the submission, but are included here as reference if it helps to see how an implementation might provide administrative interfaces for the “out of scope” areas

ResourceAccessPolicy
(assigning rights to SecAttributes)



Carol Burt
2AB, Inc.
09/28/98

Resource Administration