Using orbLock's OMG Resource Access Decision Facility (RAD) for Application Access Management

www.2ab.com

Introduction

Today, one only has to open a newspaper or turn on the television to understand that there is growing concern regarding the privacy of personal and business information. This concern is coming from both individuals and businesses, and government regulations are increasingly addressing these concerns. Although computer systems have traditionally been "secured," it is evident that future systems will be expected to enforce a higher quality and a finer granularity of security mechanisms.

One important aspect of securing future systems is the ability to control, with a fine level of granularity, the ability of individuals and systems to access information and functionality that systems provide. 2AB, Inc. has introduced the iLock™ product suite to address this problem.

In 1999 and 2000, the Object Management Group (OMG) developed a standard specification for software to address the needs of protecting fine-grained resources. This specification is titled the "Resource Access Decision Facility" and is publicly available at the OMG's web site, www.omg.org. The orbLock software that is offered by 2AB, Inc. provides a complete implementation of this specification. This paper will describe the specification and the implementation and extensions provided by orbLock™.

Security Functionality Overview

Securing distributed computing systems typically requires that security software can perform one or more of the following functions:

- Authentication
- Message Protection
- Access Control
- Auditing

Authentication

Authentication is the term used to indicate that an individual or system has proven its identity. Authentication may work in both directions in a distributed computing environment. That is, a client may be required to prove its identity to some service, and conversely, some service might be required to prove its identity to its client. The authentication function is the cornerstone for all other security functions. Without reliable authentication, all other security features that may be used are effectively meaningless.

There are various techniques used to authenticate individuals and/or systems. They include:

- Shared secret - User ID and password
- Physical tokens such as ATM cards, Smart cards...
- Digital certificates
- Biometrics such as retina scan, thumbprint readers…

Message Protection

Message protection is the term used to indicate that messages (typically in transit across networks) are protected from being viewed or modified by unauthorized persons. To protect messages from being viewed, the sending party typically encrypts messages and the receiving party has the ability to decrypt the message. To protect the message from being modified, there is typically encrypted information about the message (perhaps a hash of the message) that can only be created by the sender and read by the receiver. If this information does not match the message contents, there is a strong indication that the message has been tampered with.

Message protection techniques should almost always be used when messages travel through public networks (e.g. the Internet). Enterprises may or may not perceive the need for message protection within internal private networks.

Access Control

Access control is the term used to describe mechanisms that ensure that authenticated individuals and/or systems can only access protected resources for which they have permission. The term "protected resource" is used to indicate anything that needs protecting. It could be a computer system, some specific functionality of a system or some specific information that a system manages.

The iLock™ products specifically addresses the area of access control, and this paper is dedicated to this aspect of security and OMG standards for implementation.

Auditing

Auditing is the term used to describe the recording of information needed to detect and investigate security violations. The auditing mechanism must also provide the necessary tools to view and examine the recorded information.

Access Control Overview

Access control is the term used to describe mechanisms that ensure that authenticated individuals and/or systems can only access protected resources for which they have permission. The term "protected resource" is used to indicate anything that needs protecting. It could be a computer system, some specific functionality of a system, or some specific information that a system manages.

There are two aspects to access control. The first aspect is called "access decision." This is the aspect of access control that makes the decision to allow access or not. This decision is based on the security policy that has been associated with the protected resource. The second aspect is called "access enforcement." This is the aspect of access control that enforces the result of the access decision.

A system may have requirements to control access to a wide variety of protected resources. Some of these resources may be known to infrastructure software such as operating systems and/or communications middleware. Other resources may only be known to business logic software. This may be information such as a customer's credit history.

Infrastructure Based Access Control

Infrastructure software is software used by business software developers that handle functions common to a wide range of business functions. This would include operating systems, database management systems, communications middleware, etc. The access control that can be placed on resources known to infrastructure software is known as "Infrastructure Based Access Control."

There are many examples of resources that are typically protected by infrastructure software. Operating systems typically protect files using "read," "write" and "execute" permissions. Database management systems typically protect access to databases and database tables. Communication middleware typically protects invocation of remote procedure calls, operations or methods.

The infrastructure software is responsible for both the access decision and the enforcement aspects of access decision. Obviously, the infrastructure can use third party access control software to assist in the decision/enforcement process; however, it is ultimately the infrastructure software's responsibility. Often there are hooks that can be used by third party software to protect specific resources. For example, there are numerous products that specialize in protecting access to web pages.

Application Level Access Control

There are resources that need to be protected that only business logic can understand. For example, a system might have customer records that include general information, billing information, transaction history and credit history. Obviously, no infrastructure software will understand these resources; however, it might be necessary to control who is allowed to do what with each of these resources. It is the responsibility of the business logic to control the access to these resources. In fact, with the increasing requirements for privacy of personal information, this is becoming a very important component of the application access control requirements.

Application systems have traditionally implemented access control for business logic resources within the code of the business application. At first, this appears to be a rather easy solution; however, the fact that security policies are embedded in application code can significantly increase the long-term maintenance costs for the system. A simple change to a security policy can result in all of the costs associated with deploying a new release. Some systems may write code to make the security policy configurable; however, this typically results in different administrative procedures for different systems. When security policy is embedded in source code, it is impossible to determine the security policy without reviewing the source code to locate the specific access control logic. This makes it difficult for an auditor to determine what the policy is and whether the application software is performing authorization appropriately to meet business and legislative security policies. It is important to maintain security policies outside of source code, in a format that can be easily reviewed, to facilitate an audit process. Should problems be discovered during a review/audit, the policies can be changed without disruption to the deployed business software.

Another issue related to embedding access decision logic within the business logic is auditing. Each application must provide a means for creating audit records and a means for security administrators to view/analyze those audit records. This forces business logic developers into the business of developing infrastructure software. In addition, there is the real possibility that different applications systems will implement different auditing strategies and tools, leading to additional training and overhead for security administrators.

To address these problems, the Object Management Group (OMG) developed a comprehensive specification for standard software to manage access control for fine-grained business resources. This specification is called the "Resource Access Decision Facility." The iLock™ product, from 2AB, provides a complete implementation of this specification. The remainder of this paper summarizes this specification and 2AB's orbLock™ implementation.

OMG's Resource Access Decision Facility (RAD)

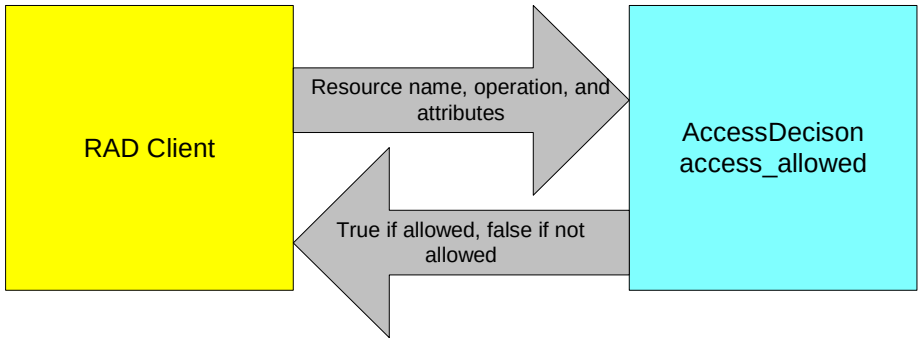
Introduction

The Resource Access Decision Facility (normally referred to as RAD) is an OMG specified service designed to assist applications that are required to control access to system resources. System resources are defined to be anything that can be assigned a name. For example, a resource might be as fine-grained as a field in a database row. On the other hand, a resource could be a collection of data centers that share some common attributes.

In order to define appropriate policy, it is necessary to define the type of access that is being requested. RAD calls this an "operation" on a resource. Common examples of operations on data element resources are "create," "read," "update" and "delete," but operations may be anything that makes sense for the specific resource. For example, a "door" resource might define operations such as "open," "close," "lock" and "unlock." RAD supports requesting access decisions for a specific operation on a resource.

RAD makes an access decision based on the resource name, the operation and one or more security attributes. Security attributes are identifying attributes of the entity requesting access to a resource. Security attributes are one of two types. Identity attributes distinguish a subject from all other subjects. A username is an example of an identity attribute. Privilege attributes are attributes that describe a property of a subject. Groups, roles and clearances are examples of privilege attributes.

A programmer invokes the "access_allowed" operation in the AccessDecision interface to request an access decision. The resource name to access, the operation and one or more security attributes are passed as arguments to this operation. RAD uses these arguments to make the decision and returns "true" if access is allowed and "false" if access is not allowed.



In the illustration above, the "RAD Client" (which is typically some service) invokes the "access_allowed" operation passing the resource name, operation and security attributes as arguments. The operation returns "true" if access is allowed, and "false" if not.

Although the functionality provided by the "access_allowed" operation is the basic functionality of the RAD service, it provides far more flexibility and functionality. In the above scenario, it is assumed that a policy engine made the access decision. RAD provides functionality for multiple policy engines to be involved in making a decision, to allow different policy engines for different resources, and to combine the results of different policy engines in unique ways.

RAD Interfaces

The RAD service consists of multiple interfaces. Some of the interfaces support functionality associated with making an access decision, and some of the interfaces support functionality associated with configuration. Those interfaces that support access decision functionality are called runtime interfaces. Those interfaces that support configuration options are called administrative interfaces.

The following runtime interfaces support access decision functionality.

- **AccessDecision** - This interface is used to invoke "access_allowed" requests that ask for an access decision. This is the interface that defines the signature of the operations that programmers may use to request access decisions.

- **PolicyEvaluator** - This interface is the policy evaluation engine of RAD. It is used to evaluate a request to access a resource. One or more PolicyEvaluators can be used to make an access decision. The iLock product is delivered with a single default PolicyEvaluator that uses the iLock Security Center service to make decisions. This default policy evaluator may be replaced and/or augmented with custom evaluators.

- **DecisionCombinator** - This interface is used to examine the results returned from one or more PolicyEvaluators and produces the final access decision. It may use simple "and" or "or" logic or it may have more complex combinatory logic.

- **PolicyEvaluatorLocator** - This is the interface that is used to locate the proper PolicyEvaluators to be used to make an access decision on a given resource name. Each resource that is to be protected can be configured to have its own set of PolicyEvaluators that are used for this purpose. Resources that are not configured for specific PolicyEvaluators use a set of "default" PolicyEvaluators.

- **DynamicAttributeService** - This interface can be implemented to examine and modify the list of security attributes that will be used to make an access decision. This is designed to allow an enterprise to develop application specific software that can modify security attributes.

The following interfaces are used to configure the operation of a RAD service:

- **AccessDecisionAdmin** - This interface is used to specify the PolicyEvaluatorLocator and the DynamicAttributeService to be used.

- **PolicyEvaluatorAdmin** - This interface is used to associate policies with resource names. A PolicyEvaluator may support one or more security policies. A resource name can then be associated with one or more policies.

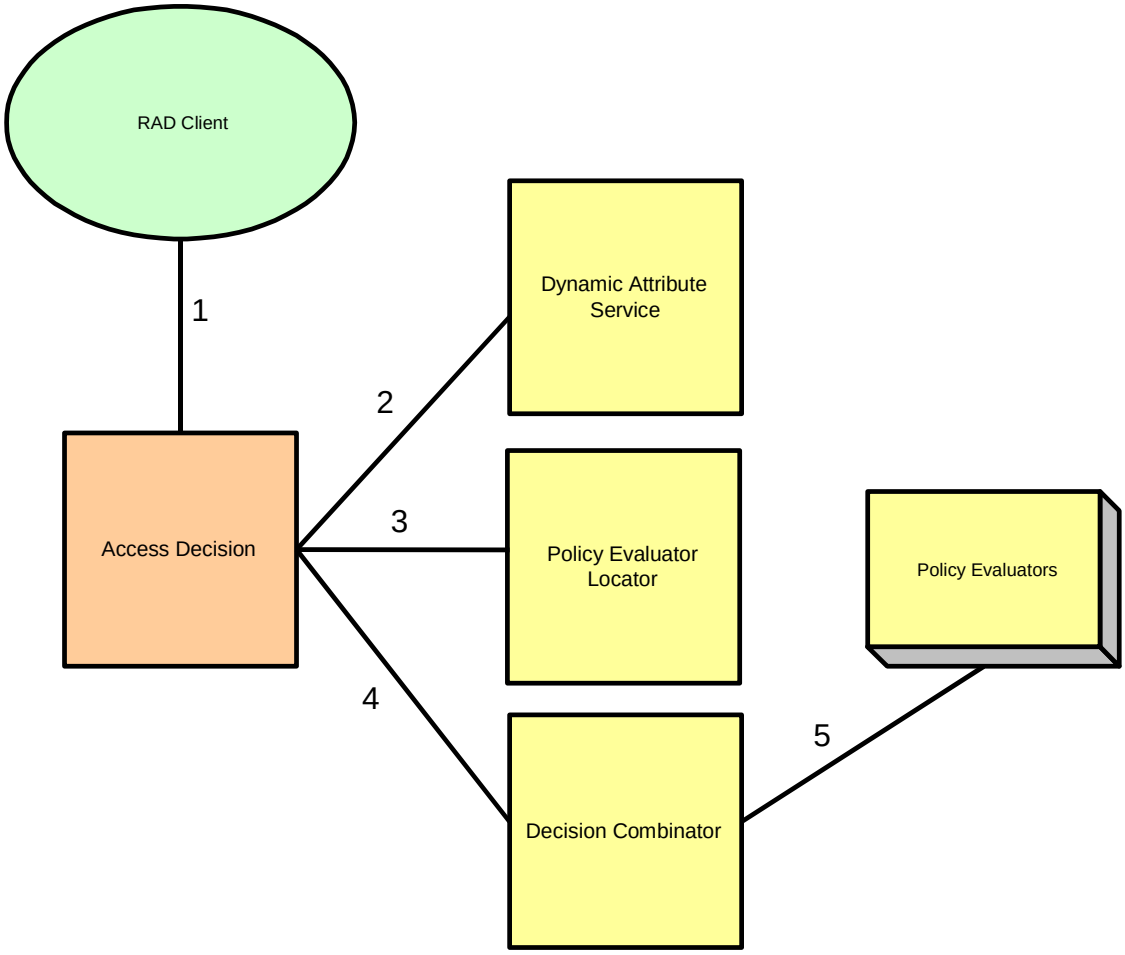
- **PolicyEvaluatorLocatorBasicAdmin** - This interface is used to set the default PolicyEvaluators and DecisionCombinator to be used for resources that are not otherwise configured to use specific PolicyEvaluators and/or DecisionCombinator.

- **PolicyEvaluatorLocatorNameAdmin** - This interface is used to associate a resource name with one or more PolicyEvaluators and/or a DecisionCombinator.

- **PolicyEvaluatorLocatorPatternAdmin** - This interface is used to associate a pattern with one or more PolicyEvaluators and/or a DecisionCombinator. A pattern has the same format as a resource name; however, the fields within the name can be regular expressions. This is used to simplify administration by allowing a group of resource names to be associated with PolicyEvaluators and/or a DecisionCombinator.

Access Decision Model

In order to understand how the RAD service functions, it is helpful to examine the basic logic involved in making an access decision. This framework is mandated by the OMG Resource Access Decision Facility to support integration of ISV policy engines as well as user customization of the facility. The following diagram illustrates the standard RAD Facility framework.



In the above diagram, the RAD client is represented in the oval. A RAD client is typically a service that has resources that it wants protected. The client uses the AccessDecision interface to request an access decision. It passes the resource name to protect, the operation and a list of security attributes to the "access_allowed" or “multiple_access_allowed” operation. This is represented by line 1. From the perspective of the client, this AccessDecision object simply returns a Boolean reply. The architecture of the RAD framework, however, supports a flexible access decision model that allows (where desired or necessary) significant customization of the service. What happens next is part of the RAD standard framework.

The AccessDecision object now consults the DynamicAttributeService, allowing it to modify the list of security attributes if necessary. This is represented by line 2.

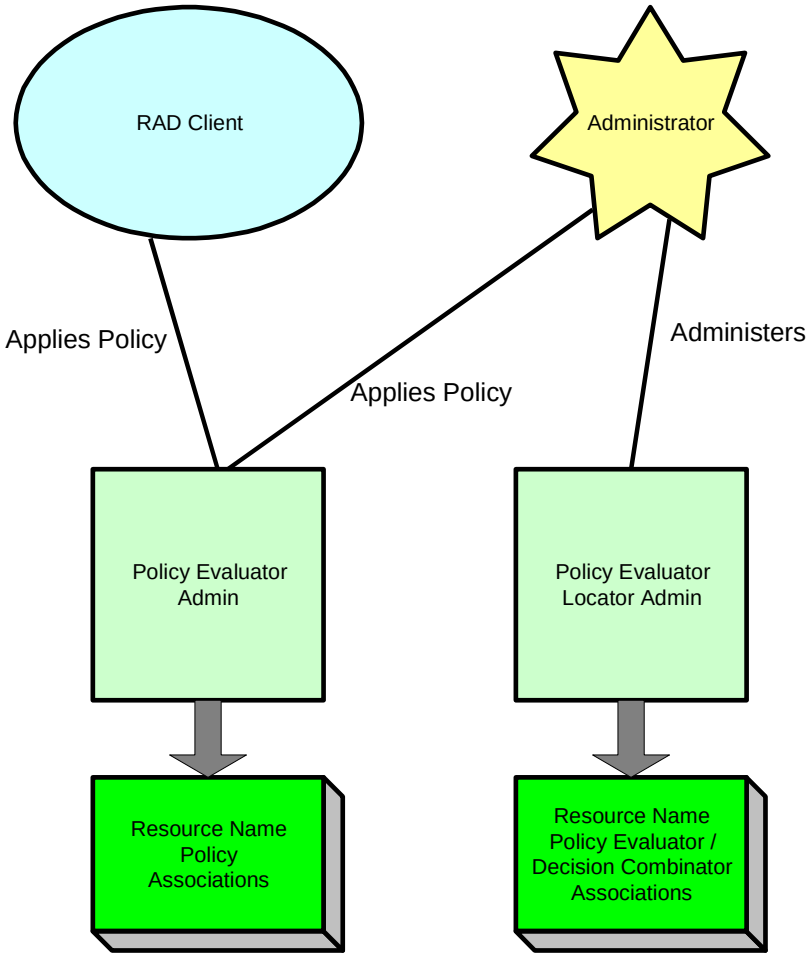
The AccessDecision object consults the PolicyEvaluatorLocator to locate the PolicyEvaluators and the DecisionCombinator that should be used for this resource name. A resource name may have PolicyEvaluators and/or a DecisionCombinator associated with the resource name, in which case it returns

these. On the other hand, there may not be any associations, and the default PolicyEvaluators and DecisionCombinator are returned. This operation is represented by line 3.

The AccessDecision object now invokes the DecisionCombinator's "combine_decisions" operation to make the access decision. The resource name, operation, security attributes and the list of PolicyEvaluators are passed as arguments to this operation. This is represented by line 4 in the illustration. The DecisionCombinator now invokes the "evaluate" operation for each of the provided PolicyEvaluators. It passes the resource name, operation and security attributes to the operation. This is represented by line 5 in the illustration. Each PolicyEvaluator returns a value indicating that access is allowed, not allowed or indeterminate. The DecisionCombinator then combines all of the PolicyEvaluator decisions and returns an access decision to the AccessDecision object, which in turn returns the decision to the RAD client.

Administrative Model

RAD provides administrative features that allow the association of resource names with PolicyEvaluators and/or a DecisionCombinator. It also provides for the association of an Access policy with a resource name. The illustration below shows this model.



In the above illustration, two entities are depicted administering the RAD service. The first is the RAD client. Typically, this service has resources to protect. For most situations, a RAD client's administration is limited to associating Access policy with a resource. This is done through the PolicyEvaluatorAdmin interface. It associates a "policy name" with a resource name. For example, a client might create a database row and assign it a resource name. It then would associate a policy name to protect that resource in the future.

The second entity that administers the RAD service is an administrator. This is a person, using some administrative tool, which administers the RAD service. They do two basic administrative tasks. The first, just as the RAD client, is to associate resource names with Access policy. This is done via the PolicyEvaluatorAdmin interface. The second task is to associate a resource name or pattern with one or more PolicyEvaluators and/or a DecisionCombinator. This is done via one of the three PolicyEvaluatorLocator administrative interfaces (PolicyEvaluatorLocatorBasicAdmin, PolicyEvaluatorLocatorNameAdmin or PolicyEvaluatorLocatorPatternAdmin).

Although it is not shown in the above illustration, it is also possible that a RAD client might want to associate PolicyEvaluators and/or a DecisionCombinator with a resource name. This can be done via one of the three PolicyEvaluatorLocator administrative interfaces.

Overview

orbLock™ provides a complete implementation of the OMG's Resource Access Decision Facility (RAD). This OMG specification provides a standardized means of managing protected resources. In addition to providing the functionality of the RAD specification, the implementation features:

- Full function default policy evaluator implementation that features:
  - Multiple rules per operation
  - Each rule can be "AND ACL," "OR ACL," "DENY ACL," "Anybody" or "Nobody"
  - Supports time constraints by date, time and day of week
  - Policy caching and notification
- GUI and command line based administration tools
- GUI testing tool for testing security policies
- Auditing for all access decisions
- Authentication interface allows securing administrative tools with any customer authentication mechanism
- Support for OMG's Naming Service, Trader Service and stringified IOR's in files
- Clients can access iLock via collocation (local method calls) or remote access
- Multiple programming interfaces
  - Standard CORBA programming
  - Java convenience classes for CORBA
  - Java iLock Interface (JII) for enterprise Java environments (with jLock product)

Product Components

The iLock software consists of multiple components that work together to support the access control function for fine-grained resources. The major components are:

- iLockRad – This is a service that implements the OMG's "Resource Access Decision Facility" specification.
- iLockRad Administration Tool – This is a GUI tool used to manage instances of the iLockRad service. This tool can connect to any instance of an iLockRad service. It manages diagnostics messages, auditing, start up options, etc. It can also manage the persistence of various objects that were specified by client programs.
- iLock Security Center– This is a service that manages persistent information about resources and security policies used by the default policy evaluator in iLockRad. This service can be shared by one or more instances of the iLockRad service and represents the domain of resources and security policies for those instances.

- iLock Security Center Administration Tool – This is a GUI tool used to manage instances of the iLock Security Center service. This tool can connect to any instance of an iLock Security Center service. This is the tool used by security administrators to manage protected resources and their associated security policies.

- iLock Security Center Command Line Tools – This is a collection of command line tools that can manage protected resources and their associated security policies. These tools would typically be used for scripting the management of resources and their associated security policies.

- Integrated Policy Testing Tool – The Security Center administrative tools include a graphical testing tool for security policies. This tool can connect to any instance of an iLockRad service. It allows a security administrator to test the results of an "access_allowed" method call. The tool allows the administrator to specify the protected resource, the operation to perform and the security attributes to test with.

Deployment options

Because the iLockRad and iLock Security Center services are distributed services, there are a wide variety of deployment options. These options should be evaluated in the context of the deployment environment and the characteristics of the application with which it is being deployed.

The iLock™ components can be deployed in one of three major ways:

- Fully Distributed – using this deployment option, users of the iLockRad service communicate with iLockRad via distributed invocations, and the iLockRad service communicates with an iLock Security Center service via distributed invocations. Using this deployment scenario, many instances of iLockRad services can share a single instance of an iLock Security Center service, thus sharing a domain of resources and security policies.

- Rules Collocated – using this deployment option, users of the iLockRad service communicate with iLockRad with distributed invocations, and the iLockRad service communicates with a collocated iLock Security Center service via direct method calls. Using this deployment scenario, each iLockRad service has it own unique iLock Security Center service and hence, its own domain of resources and security policies.

- Collocated – using this deployment option, users of the iLockRad service communicate with iLockRad via direct method calls, and this local iLockRad communicates with an iLock Security Center service via distributed invocations. Using this scenario, many iLockRad clients (with a collocated iLockRad) can share a single instance of an iLock Security Center service, thus sharing a domain of resources and security policies.

Programming Interfaces

The iLock™ software supports three programming interfaces. They are the standard CORBA programming interface, the CORBA convenience classes for Java and the Java iLock Interface (JII).

Standard CORBA Programming Interface

An application can use the standard CORBA IDL-based programming interface to use the iLockRad service. By doing this, applications can be written in Java, C++, C, Visual Basic or any language that has a supported language binding. Applications developed using this interface can utilize any functionality of the

"Resource Access Decision Facility," including building custom policy evaluators, combinators and dynamic attribute services.

CORBA Convenience Classes

Applications written in Java can use the CORBA convenience classes. Although this programming interface maintains the style of CORBA programming, it abstracts away knowledge of the location service programming (Naming, Trader), remote verses collocated invocations, and the creation of data types such as security attributes and resource names.

Java iLock Interface (JII)

Applications written in Java can use the "Java iLock Interface" (JII). This is a pure Java implementation that strips away any knowledge of CORBA. This includes any knowledge of ORB's, location services, CORBA data types, etc. The JII introduces addition functionality that allows the development of custom policy evaluators and dynamic attribute services that reside in the client process rather than in the iLockRad service. This can be a more flexible and better performance approach than running those components remotely.

Using orbLock's RAD – A Simple Use Case

In this section, we will describe a simple scenario where an enterprise wants to protect access to a collection of protected resources. For the example, we will describe a simple document management system, and the documents are the resources that we want to protect. In this simple scenario, we will assume that there will be two types of protection for reading documents. Some documents should be readable by anyone, and some should only be readable by management. When the document is created, the author will decide who may read it. In addition, the only people that will be allowed to update a document will be the author and a super user (an omnipotent system administrator). The remainder of this section will describe the steps for designing and implementing such a system.

Resource Names

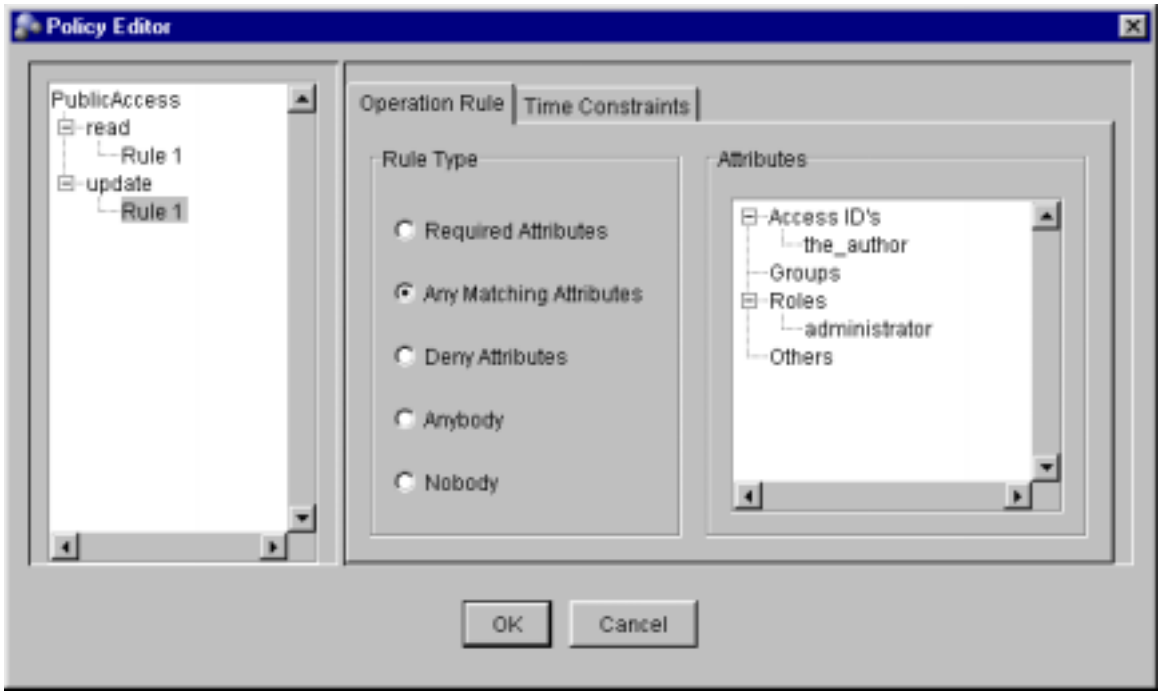
Since individual documents will be the protected resource, we must design a naming scheme for these protected resources. Resource names consist of two parts, a naming authority and a set of name value pairs. For this scenario, we will pick a naming authority of "SomeCompanyDocuments." We will have two name value pairs. They will be Author=xxx and Title=yyy. Note: the value xxx will be the access id of the author.

Security Policies

For this scenario, we will need only two security policies. The first will be for public access documents and the second will be for management access documents.

Let's name the first policy "PublicAccess." This policy should allow anyone to read a document, but only the author and the administrator may update the document. Let's name the second policy "ManagementAccess." This policy should allow anyone in management to read a document, but only the author and the administrator may update the document.

A security administrator would use the iLock Security Center administrative tool to define these two policies. The following window was used to create the "PublicAccess" policy. The selected rule for updating requires that the user must have an access id of "the_author" or a role of "administrator." It should be noted that there is no such author with an access id of "the_author." We will discuss later how this is used.



In a similar manner, the "ManagementAccess" policy will be defined to allow anyone with the role of "management" to read documents, and only the author and the administrator to update documents.

Programming the Document Management System

In the development of the document management system, the developers must utilize iLock™ in two key areas. Our examples here will use the "Java iLock Interface" (JII).

The first area is when documents are created, an associated resource name should be created, and it should be associated with a security policy. This can be accomplished with code such as the following.

```
AccessManager am = new AccessManager();  // this is done once in initialization

// this names the PublicAccess policy
String [] policies = new String[1];
policies[0] = "PublicAccess";

// this creates the resource name for the new document
 ResourceComponent [] comps = new ResourceComponent[2];
 comps[0] = new ResourceComponent("Author", "bburt");
 comps[1] = new ResourceComponent("Title", "Using iLock for Fine-grained Access Control");
 Resource res = new Resource("SomeCompanyDocuments", comps);

// this sets the policy for the resource
am.setPolicies(res, policies);
```

The second area is when documents are read or updated; iLock should be consulted to get an access decision. This can be accomplished with code such as the following to test for update.

```
AccessManager am = new AccessManager();  // this is done once in initialization
```

```
// this creates the resource name for the document being requested
 ResourceComponent [] comps = new ResourceComponent[2];
 comps[0] = new ResourceComponent("Author", "bburt");
 comps[1] = new ResourceComponent("Title", "Using iLock for Fine-grained Access Control");
 Resource res = new Resource("SomeCompanyDocuments", comps);

// test for access allowed, notes attributes obtained from a security infrastructure
boolean value = am.accessAllowed(res, “update”, attributes);
if (value == true) {
        // allow update access to the document
} else {
        // deny update access to the document
}
```

Changing Security Policy

Suppose it is decided that the security policy that determines who can update documents should be changed. This merely requires that the iLock Security Center administration tool be used to change the policy. No changes are required of the document management software to change the security policies.

Dynamic Attribute Service

You may have noted that we have not discussed how the special access id, “the_author,” is used. In the scenario described above, the author’s access id is “bburt” and not “the_author.” This is where a custom dynamic attribute service can be developed. First of all, a new Java class, that we will call DocumentsAtttributeService, will be developed. It will look somewhat like the following:

```
class DocumentAttributeService implements DynamicAttributeService {

   public SecurityAttribute [] getDynamicAttributes(Resource resource,
                                String operation,
                                SecurityAttribute [] input_attributes) {

        // The code will examine the resource, extract the author, and determine if the
        // author’s access id is in the input_attributes. If the author is in the input_attributes,
        // the access id of “the_author” will be added to the array of attributes and returned.
        // If not, the array of attributes will not be modified
   }
}
```

The system can set the dynamic attributes service by executing the following code:

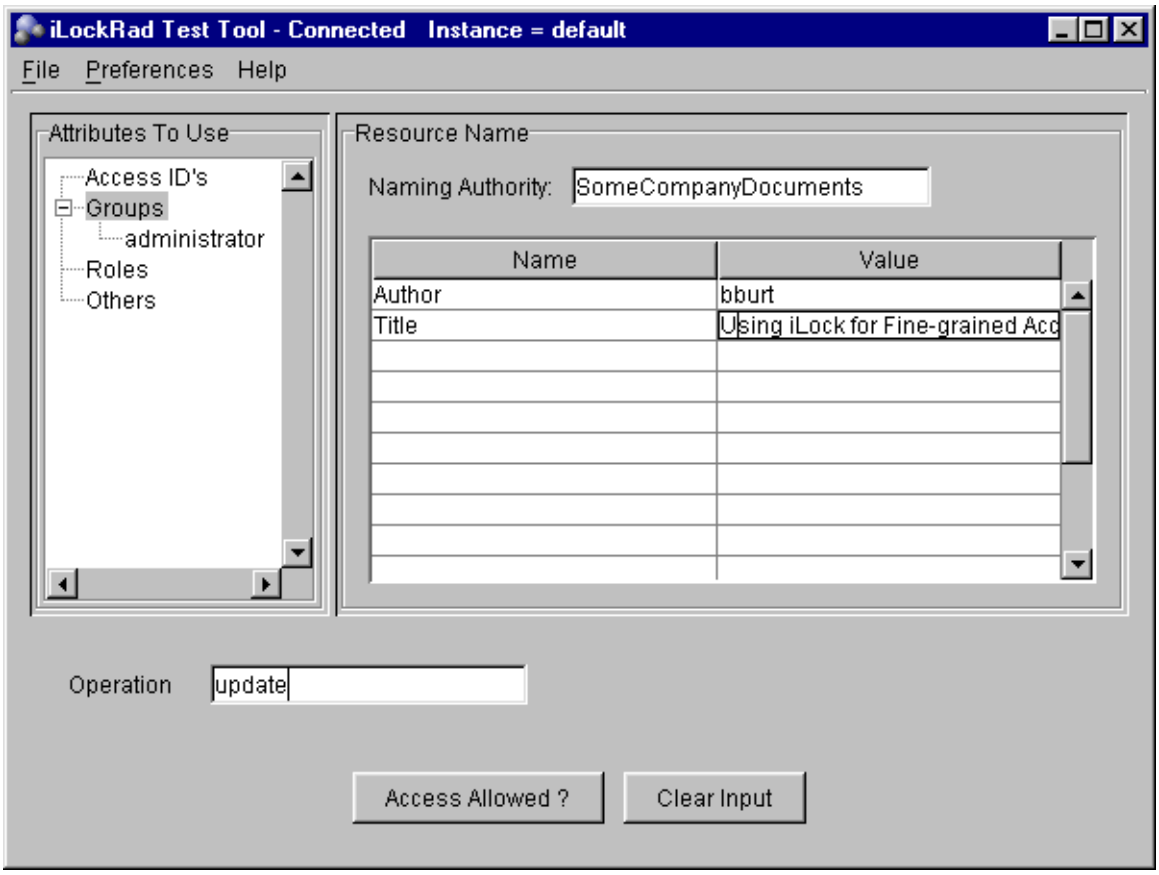
```
AccessManager am = new AccessManager();  // this is done once in initialization
am.setDynamicAttributeService( new DocumentAttributeService());
```

The iLock software will consult the “getDynamicAttributes” method each time an access decision is made, and only when the author of the specified resource is using the system will the resource be made available for updating.

By using this technique, we were able to design the system with only two security policies. Otherwise, it might have been necessary to create separate policies for each author.

Testing Security Policies

Security administrators will want a way to test security policies. The iLock Testing Tool provides a convenient method for testing security policies. For example, to test the "PublicAccess" policy described in the previous sections we might use the testing tool as follows:



Since we have defined a security attribute that has the "administrator" role, access should be allowed.

Summary

The issues surrounding the privacy of personal and business information are increasing. The "access control" aspect of security systems is of prime importance in satisfying the demands for these privacy issues.

The OMG's "Resource Access Decision Facility" specification was designed to address the issues of access control for fine-grained resources. The iLock™ product provides a robust implementation of this specification that can be used in business systems to protect resources without the need to embed security policies within the business logic. Just as iLock can be used to protect business resource, it can just as easily be used in infrastructure products to provide a robust access decision framework.

orbLock™ also includes a CORBA Security Service (CSS) that can be used to provide authentication and a source of user identity and privilege attributes. Additional information and evaluation copies of 2AB's orbLock™ software can be obtained at www.2ab.com.

For more information contact:

2AB, Inc.
1700 Highway 31
Calera, Alabama 35040

877.334.9572 (toll-free)

sales@2ab.com