A Language for Access Control in CORBA Security

Polar Humenn
Adiron, LLC
Syracuse University
CASE Center
Syracuse, NY 13244-4100
polar@adiron.com
polar@syr.edu

ABSTRACT

In this brief paper, I present the a simple straight forward formal language for describing access control based on the CORBA Security Credentials model. This language serves as a formal description of the access control mechanism. I also show how the language can be extended to describe the CORBA Security access control model that uses the notion of required rights, which is more complex, yet yields a more manageable description of the access decision. I also show that the more complex description can be reduced to the simpler form of the language and the reduction yields better performance and paths for further optimization of the access decision.

Keywords

CORBA, security, access control, distributed systems, object request broker, object oriented systems

1 INTRODUCTION

The process of granting access in any system is based on *principals* and the *resources* the principals are trying to access. Principals are entities, such as users, processes, and machines that may be authenticated. The level of authentication is a function of the underlying security system.

Common Object Request Broker Architecture (CORBA) defines a standardized distributed system based on objects[2]. CORBA distributed systems rely on CORBA Security Services[4] to provide the means necessary to authenticate principals and represent them to the application in a standard way.

To achieve better management of the access decision in system of large scale, CORBA Security introduces the notion of *required rights*. Operations of interfaces are mapped into a flat space called *rights*, which are said to be *required* for a principal to access the particular operation. A mapping from principals to rights is also provided.

In this paper I introduce the Server Access Language (SAL) that expresses the access decision logic based on the CORBA security model of credentials and an object's interface name and operation name. I then extend the language to support the required rights model and provide the transformation of the extended language to the simpler language.

Due to space restrictions placed on this workshop paper, I can only give an informal treatment to the language and its semantics. The language will be described informally using examples. The semantics of the language will also be described informally.

2 CORBA AND CORBA SECURITY

The Common Object Request Broker Architecture defines a distributed object system. Clients make requests on objects by using a middleware device called an Object Request Broker (ORB). The ORB handles data marshalling, request brokerage, communication, and in most cases generation of Application Programming Interface (API) code in common programming languages, such as C++ and Java for ease of implementation.

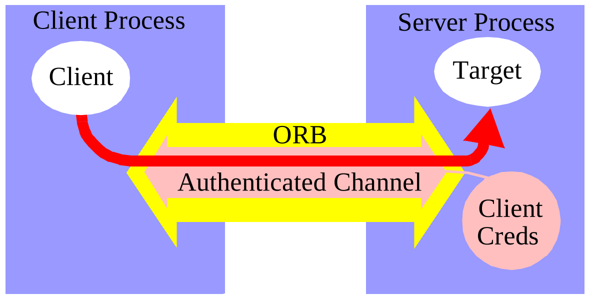

Figure 1. A Secure CORBA Request

A CORBA Request is a remote procedure call to a particular target object within a server. A Secure CORBA Request is a normal CORBA request that is tunnelled through a negotiated, protected channel. Enough information to establish this channel is exchanged between the client and the server to authenticate each other. Information about the authenticated channel is retrieved via the Credentials interface. On the server side, the Credentials represents the identity of the client.

3 CONSTRUCTS

All expressions in SAL are represented by either parenthesized structures or by names. At the top level, SAL contains declaration constructs. Declaration constructs are

(*type-tag name typed-structure*)

tuples that associate a name with a parenthesized structure so that direct substitution of the structure is possible. For example, an *attribute-family* expression has the following structure:

```
( number number )
```

A declaration for a named attribute family has the following form:

```
(AttributeFamily Corba1 (0 1))
```

Further, to illustrate substitution properties of the language, if an *attribute-type* expression has the form:

```
( attribute-family number )
```

then the expressions (Corba1 2) and ((0 1) 2) denote the same *attribute-type* construct, where the context of the encompassing expression demands an *attribute-type*. The latter expression is said to be *expanded*. When an expression is expanded fully in this manner, it is said to be in *normal form*.

Security Attribute Types

In CORBA, a security attribute belongs to a family, which is defined by a structure of two numbers, and it has a type which is defined by a family and a type number. Examples of some CORBA security attribute families and types are described by the declarations below:

```
(AttributeFamily Corba1 (0 1))
(AttributeType AccessId (Corba1 2))
(AttributeType PrimaryGroupId (Corba1 3))
```

CredentialsPred

A *CredentialsPred* is a structure that specifies a boolean query of the CORBA Credentials interface. Evaluation involves querying the Credentials for attributes of specified types and matching their value to the corresponding attribute value in the CredentialsPred structure. For example, it is common to ask the system for the client's "AccessId" attribute and compare it with a known value. The following CredentialsPred declaration illustrates this specification:

```
(CredentialsPred isBart (AccessId "bart@simpson"))
```

This declaration defines a credentials predicate, which is true if the Credentials object *has* an "AccessId" type attribute *and* that attribute's value *equals* "bart@simpson".

CredentialsPred constructs may also be composed with conjunctive and disjunctive operators, **and** and **or**, such as:

```
(CredentialsPred isBartOrHomer (or (AccessId
"bart@simpson") (AccessId "homer@simpson)))
```

The unary operator "not" is undefined for this construct. Dealing with negatives here is difficult since a negation of the (*attribute-type string*) construct would mean the negative of its conjunctive, which results in the statement, "the credentials object *does not have* an AccessId attribute *or* the attribute's value *is not equal* to "bart@simpson". This sentence is leads to problems when applied to Credentials. For instance, it is possible that the Credentials actually contains more than one "GroupId" attribute. To alleviate this apparent lack of expressiveness, negatives for access decision are pushed to the *Control*.

Controls

Controls are constructs that are evaluated against the Credentials, interface names, and operation names, and they yield the access decision, "Allow" or "Disallow". In special cases, it may yield a value of "DoesNotApply", which may arise in situations such that a control for a particular interface or operation is not defined.

In SAL, there are three types of controls: CredentialsControl, OperationControl, and InterfaceControl.

CredentialsControl

A CredentialsControl structure is a construct that is evaluated against the client's credentials and yields an access decision. A CredentialsControl structure contains a sequence of parenthesized pairs, each pair consisting of a CredentialsPred structure and a control value of either **Allow** or **Disallow**. The following description illustrates a CredentialsControl declaration.

```
(CredentialsControl OnlyBart
    ((isBart     Allow)
     (true       Disallow)))
```

Evaluation of the CredentialsControl structure proceeds by evaluating each CredentialsPred structure against the same Credentials object successively until the boolean value of "true" results. This construct resembles an if-then-else construct. So, the CredentialsControl construct of "((isBart Allow) (true Disallow))" yields "Allow" if the CredentialsPred structure associated with "isBart" evaluates to "true". Otherwise, the entire control yields "Disallow" because the predicate of the last clause is "true".

In the following sections I will continually refer to interfaces and operations defined by the following CORBA IDL definitions:

```
module test {
    interface Hello {
        void hi();
        void hello();
    };
    interface Goodbye {
        void goodbye();
    };
};
```

OperationControl

An OperationControl structure is a construct that is evaluated against the operation name and the client's Credentials. This evaluation yields an access decision. An OperationControl structure contains a sequence of pairs consisting of an operation name an a CredentialsControl structure. The following description shows an OperationControl structure declaration.

```
(OperationControl HelloBartOnly "IDL:/test/Hello:1.0"
        (("hi"       OnlyBart)
         ("hello"    ((true Allow)))))
```

The declaration is different from previous ones in that it contains an extra argument, "IDL:/test/Hello:1.0". This identifier is the CORBA interface name that is used to register the "Hello" interface with a CORBA Interface Repository, the entity in CORBA that houses the interface definition. This identifier is used at the declaration level to advise in type checking, i.e. to make sure that the operations named in the control actually are contained in the interface. A contextual restriction to the syntax of the OperationControl is that no operation name is allowed to be listed twice.

Evaluation of an OperationControl proceeds by comparing the operation name of the CORBA request with the operation names listed in the OperationControl control. If there is a match, then the associated CredentialsControl structure is evaluated against the Credentials of the request. If there is no match, the evaluation yields “DoesNotApply”.

InterfaceControl

An InterfaceControl structure is a construct that is evaluated against an interface name, the operation name, and the client’s credentials. This evaluation yields an access decision. An InterfaceControl structure contains a sequence of interface name and OperationControl structure pairs as the following declaration illustrates:

```
(InterfaceControl HelloGoodbye
    ((“IDL:/test/Hello:1.0”      HelloBartOnly)
     (“IDL:/test/Goodbye:1.0”  ((“goodbye” ((true Allow))))))
```

Evaluation of an InterfaceControl structure selecting the associated OperationControl structure by interface name and then evaluating the OperationControl structure against the operation name and the Credentials. If there is no match to the interface name the evaluation yields “DoesNotApply”.

A contextual restriction on an InterfaceControl structure is that no interface name is allowed to be listed twice.

4 ACCESS DECISION CONSTRUCT

The constructs defined above may be used to create a library of declarations. These declarations, in turn, may be used to compose an access decision policy that will be applied to a set of objects. The following construct defines the access decision function of a server.

```
(AccessDecision (InterfaceControl HelloGoodbye) Disallow)
```

There must be one and only one AccessDecision construct in a SAL description.

Evaluation of the Access Decision construct consists of an evaluation of the InterfaceControl against the Credentials of the client, the interface name, and the operation name. If the result of that evaluation is “DoesNotApply”, the access decision defaults to the value specified the third argument.

5 EXTENDING THE MODEL

The CORBA Security Specification also suggests a different model/implementation for the access decision function based on required rights. This model works on the basis that interface and operations are together grouped into a flat space known as *rights*. Therefore, one may group a set of operations on different interfaces into the same *right*, such as “get” or “set”, and then write access control logic based on those rights. Those rights are said to be *required*. This mapping is defined by a CORBA Security object called the “RequiredRights” object.

In order to write access policy based on rights, a mapping from principals to rights is necessary. This mapping is defined by a CORBA Security Administration object called “AccessPolicy”.

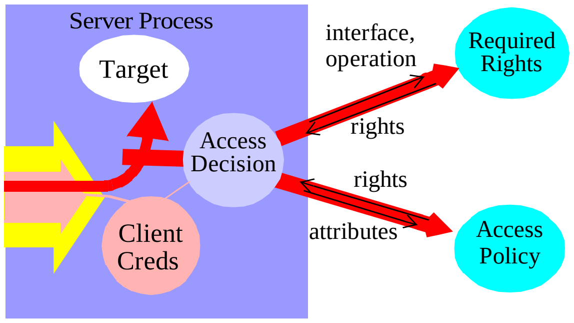

Figure 2. CORBA Access Decision Implementation

The AccessDecision object has one boolean operation, called “access_allowed”. The evaluation of this operation is defined to query the RequiredRights for the rights required for a particular interface name of the target object and the operation name of the request, and then query the AccessPolicy object for the rights granted to the client, as depicted in Figure 2. If the required rights specification is met, then the client is said to be granted access for the request and the access decision function returns “true”, otherwise the function returns “false”.

6 EXTENDING THE LANGUAGE

Extending the language to support RR/AP requires that new constructs be added to handle rights. This extended language is referred to as RRAPSAL.

Rights

One of the base constructs in RRAPSAL is the *Right* data structure. In CORBA Security, rights are defined in families similarly to the way attribute types are defined. A right has a family, which is two non-negative integers, and a value, which is a string. This notion is illustrated by the following declarations for some of the standard CORBA rights:

```
(RightFamily Corba0 (0 0))

(Right Get          (Corba0 “get”))
(Right Set          (Corba0 “set”))
(Right Manage       (Corba0 “manage”))
```

CredentialsRights

The AccessPolicy component of this RR/AP model maps the Credentials’ attributes to a list of rights. This mapping is specified by clauses containing a CredentialsPred structure and a list of Rights. The following declaration illustrates a CredentialsRights structure:

```
(CredentialsRights MyAccessPolicy
    ((isBart        (Get Set))
     (isAdmin       (Set Manage))))
```

The evaluation semantics is a bit different in that each and every clause *must* be evaluated against the Credentials. The union of the resulting rights is formed for the answer. If no CredentialsPred structure evaluates to “true”, then an empty list of rights results.

OperationRights

The OperationRights structure is a construct that specifies a mapping from on operation name to a list of rights. A descriptive example follows:

```
(OperationRights HelloRights "IDL:/test/Hello:1.0"
    (("hi"        none)
     ("hello"     (Manage Set)))
```

Like the OperationControl, this declaration contains an extra argument "IDL:/test/Hello:1.0" for the same purposes of type checking. Likewise, a contextual restriction on the OperationRights structure is that no operation name is allowed to be listed twice.

Evaluation of an OperationRights structure against an operation name produces a list of rights. If no mapping for a particular operation exists, then "NoMapping" is returned.

InterfaceRights

The InterfaceRights structure is a construct that completes the mapping of interface and operation to a list of rights. This mapping is done by specifying an OperationRights structure for each interface name. An example declaration follows:

```
(InterfaceRights MyRequiredRights
    (("IDL:/test/Hello:1.0"          HelloRights)
     ("IDL:/test/Goodbye:1.0"       (("goodbye"    Get)))))
```

A contextual restriction on the InterfaceRights structure is that no interface name is allowed to be listed twice.

Evaluation of an InterfaceRights structure against an interface name and an operation name produces a list of rights. If no mapping for a particular interface and operation exists, then "NoMapping" is returned.

Combining InterfaceRights and CredentialsRights

When using the RR/AP model, the evaluation of the CredentialsRights structure must be combined with the evaluation of the InterfaceRights structure. Evaluation of such a construct is accomplished by functional composition. Apply the evaluator of an InterfaceRights structure against the interface name and operation name. This application will produce the rights that are required. Then apply the "subset" function to the list of required rights against the list of rights produced by applying the evaluator for a CredentialsRights structure against the Credentials.

The Access Decision structure is extended for this purpose, by adding a new typed control construct, InterfaceRights-Control, as shown below:

```
(AccessDecision
    (InterfaceRightsControl MyRequiredRights
                                 MyAccessPolicy )
    Disallow)
```

The InterfaceRightsControl construct specifies that the two constructs, InterfaceRights and CredentialsRights, are to be evaluated as said above. Should the evaluation of the InterfaceRights structure result in "NoMapping", then the entire construct results in "DoesNotApply" and the specified default results.

7 EVALUATION ANALYSIS

Splitting the access control description into mappings from credentials to rights, and interface name and operation name to rights may yield easier management of access control, but its direct evaluation by separate RequiredRights and AccessPolicy objects in a CORBA security system is inefficient.

At the time that a request reaches the server, the interface and the operation are already known, and therefore, the required rights are easily known. It is better to ask if particular rights are granted instead of asking for the all rights granted to credentials and then performing a subset calculation. Unfortunately, this implemenation is precluded by using an AccessPolicy object, which only asks for all the rights granted to the Credentials.

To make the AccessDecision function more efficient, the InterfaceRightsControl construct is reduced to an InterfaceControl structure. Using an InterfaceControl over an InterfaceRightsControl construct makes the evaluation of the access decision more efficient because it does not have to evaluate *every* CredentialsPred structure that is in the CredentialsRights structure. The AccessDecision function only needs to evaluate CredentialPred structures that are associated with particular rights. Also, obtaining the entire access control description beforehand, reducing it, and evaluating the access control decision local to the server is more efficient than making calls on two remote objects and performing a combination of their results.

8 REDUCTION ANALYSIS

If both InterfaceRights and CredentialsRights descriptions are known beforehand, the two descriptions of a CredentialsRights structure (CRS) and an InterfaceRights structure (IRS) that are combined in the InterfaceRightsControl can be reduced to an InterfaceControl structure (ICS) in SAL normal form. This reduction is performed during or before server initialization.

I introduce evaluator functions for the particular structures. Partial application of an evaluator function, "eval_ic" to the ICS produces an Access Decision Function (ADF), which takes interface name, operation name, and Credentials, as the remaining parameters and returns "Allow", "Disallow" or "DoesNotApply". Partial application of an evaluator function, "eval_cr", to a CRS produces a mapping function from credentials to a list of rights, which performs the AccessPolicy object functionality. Partial application of an evaluator function, "eval_ir" to an IRS produces the mapping function from interface name and operation name to a list of rights, which performs the RequiredRights object functionality.

Partial application of the "combine" function to the partial application of the "eval_cr" function to the CRS and the partial application of the "eval_ir" function to the IRS yields the ADF as defined by the CORBA Security required rights model.

The CRS and IRS are translated immediately to an equivalent ICS that can be evaluated by the "eval_ic" function. This translation and its relation to the evaluation of the combination of CRS and IRS is illustrated by the commuting diagram shown in Figure 3.

It is the intent of the following sections to discover the "translate" function.

$$\begin{tikzcd}
\text{ADF} & (\text{eval\_cr CRS, eval\_ir IRS}) \arrow[l, "\text{combine}"'] \\
\text{IC} \arrow[u, "\text{eval\_ic}"] & (\text{CRS, IRS}) \arrow[l, "\text{translate}"] \arrow[u, "\text{eval\_cr, eval\_ir}"']
\end{tikzcd}$$

Figure 3.

9 REDUCTION TO NORMAL FORM

The translation of a description of RRAPSAL into SAL id done by first “reversing” the CredentialsRights structure and then combining it with the IRS. The IRS is a description of a function that reduces the interface and operation to a list of rights required. The idea is to convert the CRS, which is a map from CredentialsPred structures to a list of rights, to a map from individual rights to CredentialsPred structures. Once this is done, the two maps, RCS and IRS, can be combined using the “and” combinator on the CredentialsPred structures associated with each right required from the IRS.

The function “t1” represents the translation from a CRS map to a RCS map. The transformation proceeds to “reverse the map” by collecting all the CredentialsPred structures associated with each right in the CredentialsRights structure and build a single CredentialsPred structure out of them using the **or** combinator. Those associations are then collected in a list of tuples associating each right with its newly built CredentialsPred structure.

Applied to the RCS, the evaluation function, “eval_rc”, takes a list of rights and the Credentials. If a right in the RCS map is in the list of given rights, then the associated CredentialsPred is evaluated against the given credentials. The evaluation of all the selected CredentialsPred structures from the RCS map are combined using the **and** combinator.

$$\begin{tikzcd}
\text{[Right]->Credentials->Control} & \text{Credentials->[Right]} \arrow[l, "\text{f1}"'] \\
\text{RCS} \arrow[u, "\text{eval\_rc}"] & \text{CRS} \arrow[l, "\text{t1}"] \arrow[u, "\text{eval\_cr}"']
\end{tikzcd}$$

Figure 4.

To make the diagram shown in Figure 4 commute, a simple function “f1” is needed. This function simply reverses the position of the application of the credentials and the list of rights, which then results in the access decision.

To combine the diagrams, one more function “c1” is needed to bring the CRS to the access decision function. This function takes the “eval_ir IRS” evaluator function, and applies the “subset” function on it and the result of the “eval_rc CRS” function. This function makes the diagram shown in Figure 5 commute.

A transformation CRS and IRS to get an InterfaceControl structure is needed that preserves the semantics. A function called “t2” does this transformation. The transformation takes each list of rights in the IRS, which is associated with an interface and operation name, and converts it into a two clause CredentialsControl structure. The first clause is a CredentialsPred structure using the **and** combinator on all the CredentialsPred structures associated by way of the CRS with each right in the list. This first control clause is generated for the control of **Allow**. The second clause is a default clause, using the **true** predicate, with the control of **Disallow**.

$$\begin{tikzcd}
\text{ADF} & (\text{eval\_cr CRS, eval\_ir IRS}) \arrow[l, "\text{combine}"'] \\
(\text{eval\_rc RCS, eval\_ir IRS}) \arrow[u, "\text{c1}"] & (\text{eval\_cr CRS, eval\_ir IRS}) \arrow[l, "\text{f1, id}"'] \arrow[u, "\text{id, id}"'] \\
(\text{RCS,IRS}) \arrow[u, "\text{eval\_rc, eval\_ir}"] & (\text{CRS, IRS}) \arrow[l, "\text{t1, id}"] \arrow[u, "\text{eval\_cr, eval\_ir}"']
\end{tikzcd}$$

Figure 5.

$$\begin{tikzcd}
\text{ADF} & & (\text{eval\_cr CRS, eval\_ir IRS}) \arrow[ll, "\text{combine}"'] \\
& (\text{eval\_rc RCS, eval\_ir IRS}) \arrow[ul, "\text{c1}"'] & (\text{eval\_cr CRS, eval\_ir IRS}) \arrow[l, "\text{f1, id}"'] \arrow[u, "\text{id, id}"'] \\
\text{IC} \arrow[uu, "\text{eval\_ic}"] & (\text{RCS,IRS}) \arrow[l, "\text{t2}"] \arrow[u, "\text{eval\_rc, eval\_ir}"] & (\text{CRS, IRS}) \arrow[l, "\text{t1, id}"] \arrow[u, "\text{eval\_cr, eval\_ir}"'] \\
& & (\text{CRS, IRS}) \arrow[u, "\text{id, id}"'] \arrow[ull, "\text{translate}"]
\end{tikzcd}$$

Figure 6.

For example, the InterfaceRights structure,

```
(InterfaceRights MyRequiredRights
    ((“IDL:/test/Hello:1.0”          ((“hi”        none)
                                      (“hello”    (Manage Set))))
     (“IDL:/test/Goodbye:1.0”      ((“goodbye”    (Get)))))
```

and the CredentialsRights structure,

```
(CredentialsRights MyAccessPolicy
    ((isBart          (Get Set))
     (isAdmin        (Set Manage))))
```

in combination translate to the following InterfaceControl definition:

```
((“IDL:/test/Hello:1.0”
    ((“hi”        ((true Allow)
                   (true Disallow)))
     (“hello”     ((and isAdmin (and (or isBart isAdmin) true)
                                Allow)
                              (true Disallow))))
("IDL:/test/Goodbye:1.0"
     (("goodbye"     ((and isBart true) Allow)
                              (true Disallow))))))
```

Therefore, the function "translate" is a composition of the "t1" and" t2" functions and makes the diagram shown in Figure 6 commute.

A formal proof of the following theorem:

```
∀ crs irs i o c .
     (eval_ic (translate irs crs) i o c) ==(combine (eval_ir irs)
                                                  (eval_cr crs) i o c)
```

where, "i" is an interface name, "o" is an operation name, and "c" is a credentials object, is currently underway by using an automatic theorem proving tool called HOL98[1].

10 USE WITH CORBA

The use of SAL in a CORBA environment is currently used to control access to objects within a single server, although it is not precluded that servers share the same descriptions. When the server is initialized it may take a SAL or RRAPSAL description in file form, compile it to its SAL normal form, which is an InterfaceControl structure (ICS), and create the evaluator "eval_ic ICS". This evaluator is the implementation the CORBA Security AccessDecision object uses. Complex descriptions in SAL, RRAPSAL can be precompiled into SAL normal form and perhaps, be further optimized for greater performance.

Dynamic updates are performed while a server is running by modifying the SAL description and feeding it back into the server through a security administration interface. Usually, an administrative tool performs the compilation of RRAPSAL to SAL normal form before feeding it into the server, freeing the server from doing the complex transformation.

11 EXTENSIONS AND FUTURE WORK

SAL, in practice, has been extended with regular expressions for matching the values of security attributes. Constructs that query the server's credentials are also added so that more expressive credential predicates can be written, such as "the AccessId attribute of the client matches that of the server". Rights predicate structures have also been added to further generalize the concept of required rights from a straight list to an expression of rights, using such combinators as "any", "all", "and", and "or".

In future work, constructs for time and other environmental concerns may be added to the predicates. Also, SAL does not yet handle the concept of *domains* in CORBA. In CORBA, objects are said to belong to *domains*, which govern their access policy. Extensions to the controls of SAL can easily be added to support this concept. However, although the concept of domains exists the CORBA Security Specification, it is not yet sufficiently specified to add to the SAL language. This situation will change when the OMG Domain Management specification is completed[5].

12 CONCLUSION

I have presented a language for access control for the CORBA security credentials model based on interfaces and their operations. The language expresses non-trivial access control policies by defining predicates on a client's credentials. Negatives can also be expressed using the **Disallow** control. A description written in SAL normal form along with the evaluator "eval_ic" serves as an implementation of the CORBA Security AccessDecision object, which is the entity that controls the access decision of the request on the server side.

It also has been shown that the descriptive power of the CORBA Security Required Rights/Access Policy (RR/AP) model is supported by an extension to SAL, called RRAPSAL. The RR/AP model has its benefits that it may yield a more manageable description; however, it lacks some expressiveness in that the semantics of RR/AP does not allow for the introduction of negative controls.

It has been shown that the reduction of RRAPSAL to SAL normal form results in a more efficient evaluation than that of straightforward RRAPSAL. It is also obviously faster that evaluation of the RR/AP CORBA object model, which must make remote calls on RequiredRights and AccessPolicy objects. Initializing servers with complete descriptions for RR/AP in RRAPSAL compiled down to SAL normal form makes the evaluation of access decision more efficient.

Having descriptions of access control in a formal language instead of implications from queries on object interfaces such as RequiredRights and AccessPolicy opens up the door for semantic tools that can help with creating a good access decision policy by formally analyzing policy and eliminating its deficiencies.

ACKNOWLEDGEMENTS

I would like to thank Dr. Shiu-Kai Chin, Fred Dushin, and Nick Battle, for their comments on earlier drafts of this paper, and Thumrongsak (Tan) Kosiyatrakul for his continuing help with the HOL theorem prover.

REFERENCES

1. Gordon and Melham, *Introduction to HOL*, Press Syndicate of the University of Cambridge, ISBN 0 521 44189 7, 1993.
2. Karjoth G., *Authorization in CORBA Security*, Fifth European Symposium on Research in Computer Security, Springer-Verlag, Lecture Notes in Computer Science 1485, p. 143-158, September, 1998.
3. Object Management Group, *CORBA 2.3*. Document formal/98-12-01, December 1998.
4. Object Management Group, *CORBA Services*, Document formal/98-12-17, Chapter 15, December, 1998
5. Object Management Group, *Domain Membership RFP*, Document orbos/98-11-24, November, 1998
6. Pierce B., *A Taste of Category Theory for Computer Scientists*, CMU-CS-88-203, Carnegie Mellon Univ, 1988