1. Appendix - Security Guidelines

The COAS interfaces may be used in many different environments with widely varying security requirements that range from no security to extreme security. For this reason the COAS interfaces do not expose any security information. COAS relies on the underlying CORBA infrastructure and services which provides all the security mechanisms needed without exposing it in the interfaces.
An attribute of security that of concern to many people is to maintain confidentiality of certain (sensitive) information about them. For COAS this implies being able to filter requests by:

- who is accessing the information,
- who the information is about,
- what information is being accessed.

Other common security concerns could be preventing unauthorized modification of data, tapping into communications to acquire sensitive information, and causing loss of service by over burdening a service. CORBA Security provides robust mechanisms to address these and other concerns. Some of the security properties it does deal with includes authentication, authorization, encryption, audit trails, non-repudiation, etc. CORBA Security, in its default mode allows these security concerns to be addressed without the client and server software being aware of it. This is a powerful notion, allowing security policies to be created and enforced after applications and systems have been created and installed.
Other CORBA and CORBA Security features provide mechanisms for applications to extend these security capabilities. For example they can obtain credentials from the ORB and implement filters that can look at specific data passed to and returned from operations.
It is a requirement of the COAS to provide confidentiality of information that is stored about an individual. This requirement fuels the need for fine grained access control on clinical observations that are associated with subjects of observations.

1.1 Security Requirements

For the COAS to be secure in its possible dissemination of information it needs to adhere to several requirements:

- The COAS needs to authenticate a client's principal identity, role, affiliation and other security attributes.
- The COAS needs to transmit information confidentially and with integrity.

The first requirement states that the entire COAS interface implementations must be able to identify a potential client. If it cannot authenticate a client, then the client may be severely limited in the particular requests that the COAS can service. The CORBA Security Service provides the mechanisms for a server to authenticate a client.
The second requirement provides for the confidentiality of the information. The client must communicate with the COAS using not only encryption to protect data, but signature as well, so as not to have data tampered with during communication. There is no sense in putting a Sensitivity level of "OwnerOnly" on an observation and have its value transmitted to the owner in the clear. The CORBA Security Service provides these capabilities, including SSL.
The problem is, How does one get CORBA to support this access policy model?

1.2 CORBA Security Overview

In an effort to keep the COAS interfaces security unaware, i.e. no extra visible security relevant parameters in methods, access policy must be adhered to from behind the interfaces. The CORBA security model offers several ways to apply security policy to method invocations.
The CORBA Security Specification (CORBAsec) is not a cookbook for using CORBA security in building applications. It is a specification of a general framework with which ORB vendors and application vendors can build a multitude of different security policy models. The CORBAsec also gives the interfaces for

which implementations of applications can access those security services that are supplied with a secure ORB.
A secure COAS implementation that can control access to specific observations must be aware of the security services offered by the ORB. This caveat also means that a client's ORB may have to know the kind of ORB and the security services that is used by the COAS.
The CORBA security specification outlines a general security policy model. Although the specification is vague about which approach should be taken, it is specific enough to be able to choose from a couple of models that can be supported.
The CORBA security model bases itself on credentials and security domains. Credentials are data objects that contain attributes such as privileges, capabilities, and sensitivity levels, amongst others. Security domains are mappings from credentials to access rights. Credentials can be encrypted and signed to prevent tampering and to achieve a level of trust between client and server. CORBA credentials get passed with requests beneath the visible level of the interface. CORBA security services give the clients and servers the ability to authenticate/verify credentials to implement policies in security domains.
Many different schemes, algorithms, services, and vendor implementations exist to provide implementation of security policy, and many different implementations of those schemes may be integrated into a CORBA compliant ORB. It is not the purpose of this specification to dictate the specific implementation of an ORB and security services that should be used, but to outline the external requirements for the COAS implementation. These requirements and guidelines aid in selecting a secure ORB and the level security functionality needed to implement the COAS access policy model.

1.3 Secure Interoperability Concerns

CORBA has built the communication bridge between distributed objects creating a interoperable environment that spans heterogeneous platforms and implementations. However, security adds another layer of complexity to the issue of interoperability. ORB implementations are neither required to include security services nor required to provide an interoperable mechanism of security services. However, a specification does exist for the target object to advertise, via the IOR, the security services that it supports and the services it requires from the client. Both the client and server ORBs must use compatible mechanisms of the same security technology.
The CORBA Common Secure Interoperability (CSI) Specification defines 3 levels of security functionality that ORBs may support. The levels are named, CSI Level 0, CSI Level 1, and CSI Level 2. Each level has increasing degrees of security functionality.
The CSI Level 0 supports identity based policies only and provides mechanisms for identity authentication and message protection with no privilege delegation. The CSI Level 1 adds unrestricted delegation. The CSI Level 2 can implement the entire CORBA Security Specification at Security Level 2.
Each CSI level is parameterized by mechanisms that can support the level of security functionality, such as SPKM for CSI Level 0, GSS Kerberos for CIS Level 0 or CIS Level 1, and CSI_ECMA for CSI Level 2. Future developments in security functionality and mechanism are not restricted, and mechanisms can be added to each level.
The ORB implementations may use different security technology with differing capabilities and underlying mechanisms, such as SSL, DCE, Kerberos, Sesame, or other standards. Choosing the ORB and its underlying security services will be critical to protecting COAS, and it will influence the implementation of the access policy that a secure COAS implementation must support.
For example, an ORB that only supports SPKM, i.e. CSI Level 0, can only authenticate clients and provide confidentiality and integrity of communication. It cannot support definition and use of security attributes beyond an access ID. Support for security attributes beyond an access ID require CSI Level 2. Therefore, using an ORB that only provides CSI Level 0 will require the COAS to maintain its own information on the credentials of clients.
Even if an ORB's security technology supports the definition of security attributes that can be delivered to the COAS, i.e. CSI Level 2, there are still concerns involving the trust between the client and the COAS.

1.4 Trust Models

The available trust models for the COAS is simplistic. Since the COAS is a communications end point and does not require to make requests on other services on a client's behalf, a delegation trust model is not needed. This simplifies the model and eliminates an absolute need for a CSI Level 1 or CSI Level 2 secure ORB (although they may use them).
There are two basic trust models for the COAS. If the COAS and its client are implemented using CSI Level 0 or CSI Level 1 ORBs, only the first trust model can be supported. If a CSI Level 2 ORB is used, both trust models can be supported. The trust models are:

1. The client's identity can and is trusted to be authenticated. However, the client is unable or untrusted to deliver the valid credentials.
2. The client is trusted to deliver the correct credentials.

In the first model, the client ORB is required to authenticate its principal (the user) and provide authentication information to the server ORB. The methods used to accomplish principal authentication is specific to the mechanisms (e.g. DCE or Kerberos) that the selected ORB supports. Management of those identities is also specific to the mechanism. The server ORB must have a compatible mechanism that verifies the authentication information and carries out mutual authentication of the client.
With this trust model, a secure COAS implementation must maintain and manage a map of identities to privilege attributes. CSI Level 0, 1, and 2 ORBs are able to support this trust model.
Even if the ORB has CSI Level 2 functionality, it may be a local policy that a COAS does not trust the credentials brought forth from an authenticated client. In that case, the COAS must maintain the map or use a default set of security attributes for requests from clients it does not trust.
In the second model, the client ORB is required to authenticate its principal and acquire its valid credentials. The methods used to accomplish principal authentication and acquisition of privilege attributes are specific to the mechanism that the selected ORB supports, such as DCE and Sesame. Management of those identities and attributes are also mechanism specific. A secure COAS installation using this trust model must take a careful look at that management scheme and operation, evaluate it, and decide to trust it. In such a scenario, the server ORB, which has CSI Level 2 functionality, automatically verifies the credentials on invocation.
A secure COAS built to the second model leaves management of identities and their attributes to the security services policy management system used by the ORB. The COAS may manage security attributes for the data itself.
A secure COAS built to the first model will have some scheme to manage trusted identities and their credentials. There is no interface or plan in the COAS to specify this kind of management.

1.5 CORBA Credentials

To adhere to the credential model that supports trait specific access policy, a set of credentials must contain privilege attributes such as the identity of the client, the role in which the client is actively represented, and the sensitivity level of information to which the client is allowed access. It will be the responsibility of a COAS implementation to advertise to potential client vendors the specifics of these attributes and how to represent them externally. A client ORB needs to ascertain certain credentials about the user and must pass them to the COAS. An external representation of those credentials is needed so that credentials can be passed between client and server within the CORBA security services. The CORBA Security module defines the structure for this representation.

```
module Security {

    const SecurityAttributeType   AccessId  = 2;
    const SecurityAttributeType   Role      = 5;
    const SecurityAttributeType   Clearance = 7;

    struct SecAttribute {
        AttributeType      attribute_type;
        Opaque             defining_authority;
        Opaque             value;
```

```
    };
    typedef sequence<SecAttribute> AttributeList;
}
```

Listed above are the relevant pieces of the specification from the Security module that apply to externalizing credential information.

1.6 CORBA Security Domain Access Policy

In addition to a credential based scheme, CORBA defines security domains. The purpose of this section is to explain and illustrate the use of the standard CORBA security policy domain and the way in which it may be used to implement a security policy for the COAS. This section offers a recommendation to a COAS implementor in order to give a feel for the kinds of security policy a COAS implementation may need to support. It should also guide the implementer in evaluating a secure ORB and available security services.

A security domain governs security (access) policy for objects that are managed within that domain. In order to make scalable administration of security policy, these domains map sets of security credentials to certain sets of rights. A right is a sort of an internal security credential.

CORBA defines a standard set of rights that are granted to principals within a security domain. A security domain administrator manages that map through the SecurityAdministration module's DomainAccessPolicy interface. Access decision then can be based on a set of required rights and the rights granted to the client by the domain access policy, by virtue of the client's credentials.

ORB security service vendors will supply a security policy management infrastructure that implements the standard CORBA rights scheme. The COAS must use security services that can place different required rights on the COAS interfaces. Some ORB security services may allow a security domain to create special rights. However, CORBA defines a standard set of rights: get, set, and manage. This right set will suffice to handle the COAS.

In the model just described there is one security domain for all of the COAS components. The CORBA rights families scheme within a single security policy domain suffices to differentiate the security nature of the methods. More generally any number of domain models can be used, such as a separate security domain for each COAS component.

1.7 Request Content Based Policy

The CORBA standard domain access policy scheme only protects methods from invocation at the target and without regard to content of the request. The COAS needs a more fine grained access control in order to implement the content based access policy required (e.g. access policies for different observations). The COAS implementations must be made security aware to implement an access policy based on the value of arguments in a request. There are multiple ways to implement this policy using a secure CORBA implementation.

The CORBA Security Specification supplies two different schemes represented by an interface hierarchy, which are Security Level 1 and Security Level 2 (these should not be confused with CSI Levels 0, 1, and 2). These interfaces describe the level of security functionality that is available to security aware implementations.

Security Level 1

For the COAS to take advantage of CORBA security in order to implement its access policy model, the ORB must at least implement the CORBA Security Level 1 interfaces. A Security Level 1 compliant ORB supplies an interface to access the attributes of the credentials received from the client.

Using the SecurityLevel1 interfaces, which is simplistic, gives the implementation of the COAS interfaces the ability to examine the client's credentials and compare them to the access control information that is managed as the access policy. However, the implementation of the COAS must be security aware.

```
module SecurityLevel1 {
```

```
    Current get_current();

    interface Current {
        Security::AttributeList get_attributes(
                in  Security::AttributeTypeList attributes
        );
    };
}
```

Using the Security Level 1 interfaces, each implementation of a COAS query interface must call the get_attributes() function on the Current pseudo object, examine the attributes, compare to the access policy information, and make the access decision. If COAS implementation chooses to use Healthcare Resource Access Decision Facility, it constructs an appropriate resource name, operation name, and passes them to ResourceAccessDecision::access_allowed() along with the attributes received from Current::get_attributes(). Details on how COAS implementations must use HRAD are provided in Section 1.8. In latter case COAS does not need to examine the attributes and implement any access decision logic. The COAS implementation should enforce the access decision according to the semantics of the particular COAS operation. It is the responsibility of the client's ORB to acquire the proper credentials securely. It is the responsibility of the server's ORB to authenticate credentials received from the client, extract the security attributes from them, and make them available to the implementation through the Current::get_attributes() method.

Security Level 2

Using an ORB which supplies the Security Level 2 interfaces, the implementation can be somewhat free of making the access control decision in the implementation of the query interfaces. Having an implementation that is security unaware is attractive in CORBA, because security policy decisions can be made underneath the functionality, and they have the ability to be changed without retooling the application.

As with any framework, there are several ways in which to use the Security Level 2 interfaces. One approach could be to implement a replaceable security service for the access decision. Security Level 2 describes a method in which security can be enforced by the creation of an Access Decision object. The AccessDecision object would interact with a DomainAccessPolicy object to get effective rights and compare those to rights returned from the RequiredRights interface.

Some secure ORB implementations may allow the installation of specialized Access Decision objects to be used in conjunction with specialized DomainAccessPolicy objects. In the Security Level 2 interfaces, the specification implies access control only on the invocation of a method and not the contents of the request.

```
module SecurityReplaceable {

    interface AccessDecision {
        boolean access_allowed (
            in   SecuirytLvel2::CredentialList   red_list,
            in   CORBA::Object                    target,
            in   CORBA::Identifier               operation_name,
            in   CORBA::Identifier               interface_name
        );
    };
}
```

Currently, the AccessDecision object specified in the SecurityReplaceable module does not take the invocation Request as an argument. It only makes an access decision based on the credentials received from the client, the target object reference and operation name, and the target's interface name. This criteria is insufficient to implement the content based access policy, if needed by a COAS implementation to be automatically performed by the ORB.

Since the COAS requires access control on the contents of the method invocation (such as asking for the value of the HomePhone trait), this scheme of replacing these Security Level 2 components cannot be used. ORB security services that use the standard CORBA domain access policy may use third party implementations for these components. This standard domain access policy functionality gives a the COAS a high level of invocation protection that is orthogonal to the content based access policy. Some COAS

need the standard domain access policy functionality in addition to providing content based access policy. Therefore, another approach must be taken.
A content based access policy can be implemented in a Security Level 2 ORB by using an interceptor. A request level interceptor takes the Request as an argument and therefore, it can examine the content of the invocation arguments.

```
module CORBA {

    interface Interceptor { ... };
    interface RequestLevelInterceptor : Interceptor {
        void client_invoke( inout Request request );
        void target_invoke( inout Request request );
    };
}
```

Installing an interceptor on an ORB is ORB implementation specific, and each ORB vendor may have their own flavor of interceptors. The ORB calls the request level interceptor just before the invocation gets passed to the server implementation by using the target_invoke() operation. The interceptor uses the Dynamic Skeleton Interface (DSI) to examine values of the arguments of the invocation and make access decisions. These access decisions are also based on the credentials received from the client and the access policy. The interceptor will deny access to the invocation by raising the an exception. The server's ORB will transmit this exception back to the client.
The use of the interceptor scheme frees the implementation of the COAS interfaces from the implementation of the access decision policy. If the access policy model changes, then the interceptor can be changed out without retooling the COAS implementation.
As awareness of the need for more powerful and flexible security policy management grows, more tools to create, manage, and analyze policy will come into existence. A COAS implementation relying on interceptors to implement its security policy may be able, with relative ease, to switch to using more robust policy services as they become developed.

1.8 Use of Healthcare Resource Access Decision Facility

"Resource names used for obtaining access decisions from HRAC facility by COAS-compliant services, should be created in a predefined manner:

COAS_HRAC_Resource_Name ::=
'IDL:omg.org/DsObservationAccess' +
{‘ObservedSubjectId’, <ObservationData.observed_subject_id> } +
{‘QualifiedCodeStr’, <Stringified ObservationData.observation_type>}+
{‘TimeSpanStr’, <Stringified ObservationData.observation_time>}+
[{‘ObservationId’, <ObservationData.observation_id>}]

Text below explains the expression above in English.
If a COAS-compliant service uses Healthcare Resource Access Decision facility (HRAD), it shall:

- create HRAD resource names according to the following rules:
    1. "resource_naming_authority" data member of ResourceName shall adhere to the syntax of NamingAuthority::AuthorityIdStr type. For the corresponding datum element of type AuthorityId, the value of authority shall be 'IDL'. The value of naming_entity shall be 'omg.org/DsObservationAccess'.
    2. First element of ResourceName data member resource_name_component_list is mandatory. Its *name_string* shall have value of 'ObservedSubjectId', and the value of value_string shall be the value of *observed_subject_id* data member of the corresponding datum element of type ObservationData for the observation to be accessed.
    3. Second element of ResourceName data member *resource_name_component_list* is mandatory. Its member *name_string* shall have value of 'QualifiedCodeStr', and the value of *value_string* shall

be the stringified, via TerminologyServices::TranslationLibrary.qualified_code_to_name_str(), value of *observation_type* data member of the corresponding datum element of type ObservationData for the observation to be accessed.

4. Third element of ResourceName data member resource_name_component_list is mandatory. Its *name_string* shall have value of 'TimeSpanStr'. The value of the corresponding name_string data member shall be the stringified, via [EDITOR: PUT HERE THE NAME OF THE OPERATION USED TO STRINGIFY TIMESPAN VALUES], value of *observation_time* data member of the corresponding datum element of type ObservationData for the observation to be accessed.
5. Fourth element of ResourceName data member resource_name_component_list is optional. If it provided, its data member *name_string* shall have value of 'ObservationId'. The value of the corresponding name_string data member shall be the value of 'observation_id' of the corresponding datum element of type ObservationData for the observation to be accessed.

- Create HRAD operation name according to the following rules:
  1. When serving invocations of operations that semantically mean "get", operation in DfResourceAccessDecision::access_allowed() shall have value 'read'.
  2. When serving invocations of operations that semantically mean "set", operation in DfResourceAccessDecision::access_allowed() shall have value 'write'.
- Obtain security attributes of the invoking principal via SecurityLevel1::Current.get_attributes() (read Section 1.7 for details) or other means.
- Obtain resource access decision(s) by invoking either access_allowed() or multiple_access_allowed() on DfResourceAccessDecision::AccessDecision interface.
- Enforce the decision according to the semantics of the operation the COAS-compliant service is serving.
- It is not mandated by this specification how exceptions caught during an attempt to invoke either access_allowed() or multiple_access_allowed() on DfResourceAccessDecision::AccessDecision interface are handled by COAS-compliant service."