Concepts of the Security Model

In addition to the standard MDI development patterns, which are described in E2E's White Paper on Model Driven Integration, three concepts form the basis for E2E's MGS framework: the **Principal Object**, **Role Based Access Control** and the **InterceptorModel**.

On this Page:

Related Pages:

- Security Reference

Example File (Builder project Advanced Modeling/Security):

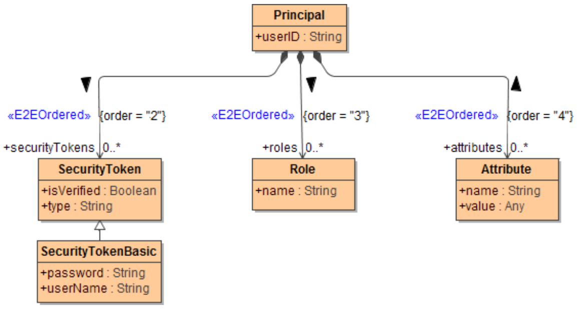
<your example path>\Advanced Modeling\Security\uml\interceptorHelloWorld.xml
<your example path>\Advanced Modeling\Security\uml\interceptorHelloWorldAdvanced.xml

The Principal Object

The **Principal** object represents the user invoking the current request. Besides a compulsory attribute **UserID**, it may have one ore more security tokens, roles, and attributes.

- **UserID** is a unique identifier of the user. Depending on the authentication process it may or may not be identical with a user name.
- **SecurityTokens** is an array of **SecurityToken** objects describing the credentials used for authentication. The attribute **Type** defines what mechanism of authentication was used and how the rest of the object is structured. Currently, only **Basic** is supported, which works with plain text user name and password. The attribute **IsVerified** is set to true, if the given credentials were successfully verified.
- **Roles** is an array of **Role** objects. See the next chapter for more information about role based access control.
- **Attributes** is an array of **Attribute** objects. An attribute is a name-value pair that can be used to specify any additional information about the Principal not covered by security tokens and roles.

Figure: Class Diagram of Principal

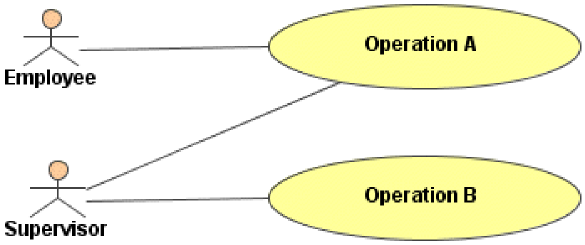


If authentication is not configured, the Principal will have the UserID **anonymous** with no security tokens, roles, and attributes assigned.

Role Based Access Control

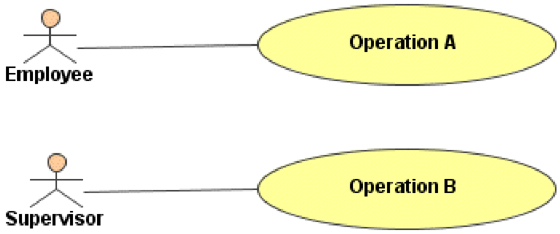
Various job functions within a given organization have particular roles assigned, granting the permission to perform certain operations. Since users are not assigned any rights directly, but acquire them through their role (or roles), management of individual user rights becomes a matter of assigning the appropriate roles to the user, simplifying common operations such as adding a user, changing a user's department or job function.
Besides explicit manual role assignment, there may also be dynamic, rule based assignments to determine the roles a user currently has, e.g. by mapping user credentials or by using system information like the current IP address.
Within the E2E Bridge, the roles allowed or required to access certain operations are modeled within UML Use Case diagrams which are relevant for the E2E Model Compiler. Each actor accessing an operation represents a role with specific execution rights. Users invoking a certain operation need to have at least one of the required roles as defined by the actors. In the following paragraphs, we will investigate several examples to describe how the E2E Bridge manages roles and access rights.
In Example 1 (see Figure 2), Operation A can be accessed by both user roles, Supervisor and Employee, whereas Operation B is only accessible by users with the role Supervisor.

Figure: Use Case and Roles Example 1



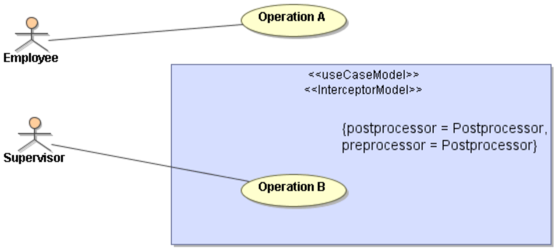
In the first example, operation **A** can be accessed by both, **Supervisor** and **Employee**, whereas operation **B** is only accessible by users with the role **Supervisor**.

Figure:Use Case and Roles Example 2



In the second example, operation **A** is only accessible with the role **Employee**. A user explicitly needs the role **Employee** to access operation **A**, even if he is a **Supervisor**.
In the Bridge, access rules are only enforced if explicitly enabled by the modeler. This is done by putting protected operations into a System Boundary of stereotype <<InterceptorModel>>.

Figure Use Case and Roles Example 3



In example 3, Operation A is accessible by anyone and the actor Employee is just a symbolic representation of any user invoking the request. Operation B however is protected by role based access control and a user requires the role Supervisor to invoke it.

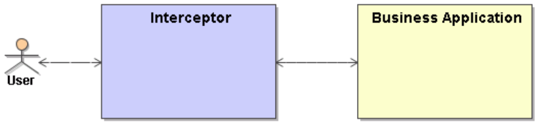
Interceptor

Any security measure like authentication, authorization, encryption and validation could be implemented as part of an application. However in practice it is preferable to make this functionality transparent for applications and developers.
Advantages of a separation are:

- A proven security infrastructure and authorization model can easily be applied to new services.
- The security infrastructure and authorization model can be updated or replaced without impact on the business logic.
- Security is a complex topic and does require special knowledge. With separation application developers can concentrate on their area of expertise.

A common concept is the **Interceptor**. An Interceptor is an application and/or device which is sitting in front of the actual service and intercepts any request sent to and reply sent from the business application. The interceptor can also be used without security - e.g. to implement central logging or validation using the preprocessor. In that case one define a use case model without actors (roles) but having pre- and postprocessors defined.

Figure: Concept of a Basic Security Interceptor

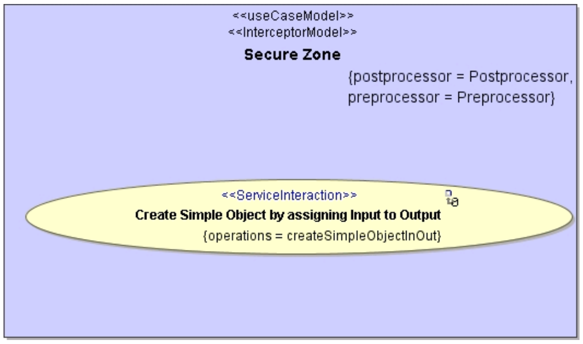


The functionality of an interceptor varies. It typically covers one or more of these areas:

- **Authentication** of users
- **Authorization** of users
- **Encryption** and decryption of data
- Creation and verification of **signatures**

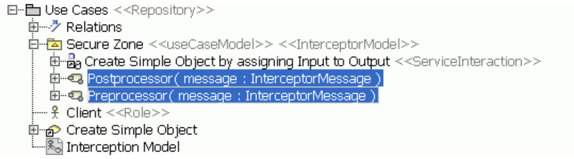
Common network components like firewalls, secure-proxies, authentication modules for web servers fit above definition. There are 3rd party products for transparently protecting WebServices that can be combined with the E2E Bridge to implement complex security requirements.
E2E Bridge offers its own, built in Security Interceptor mechanism.
The interceptors are activity diagrams that are assigned to the pre- and postprocessor tagged values of the interceptor model (see also chapter Role Based Access Control):

Figure: Interceptor activities are referred to in the <<InterceptorModel>>


<<useCaseModel>>
<<InterceptorModel>>
Secure Zone
{postprocessor = Postprocessor,
preprocessor = Preprocessor}
<<ServiceInteraction>>
Create Simple Object by assigning Input to Output
{operations = createSimpleObjectInOut}

Typically, these activities are put into the interceptor model:

Figure: Interceptor in the containment tree

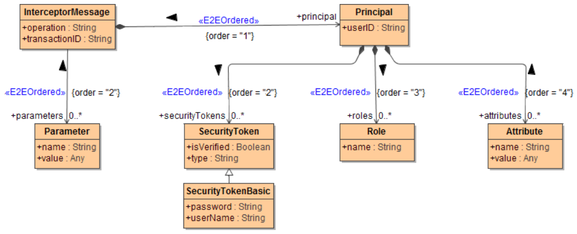


```
Use Cases <<Repository>>
  Relations
  Secure Zone <<useCaseModel>> <<InterceptorModel>>
    Create Simple Object by assigning Input to Output <<ServiceInteraction>>
    Postprocessor( message : InterceptorMessage )
    Preprocessor( message : InterceptorMessage )
  Client <<Role>>
  Create Simple Object
  Interception Model
```

> The root package (**Use Cases** in the example above) and sub-packages containing an interceptor have to have the stereotype <<Repository>> applied.

Deprecated: In former releases, the pre- and postprocessor had one input object of type **Preprocessor** respectively **Postprocessor**. However, they are deprecated, because casting the input- and output messages proved to be error prone.
Thus, we recommend to use as only in- and output parameter of interceptor activity diagrams the **InterceptorMessage**:

Figure: Interceptor message

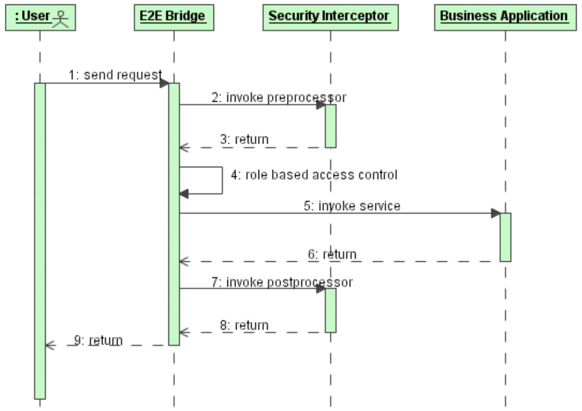


The **InterceptorMessage** contains name of the intercepted operation and a reference to the **principal**. Note, changes to the parameter **operation** are ignored. The only difference between the interceptor message and the deprecated pre- and postprocessor message is the way the operation parameters are being passed. In the latter case, an artificial wrapper class had to be designed and the Input- and Output messages had to be cast to this wrapper. However, now the interceptor message holds the given parameters explicitly in the **parameters** array. This array contains **Parameter** objects containing the parameter name and its value. If the modeller wants to use the parameter value, he must typically cast it to the parameter type as given in the signature of the operation (an example can be found in chapter **Modifying Output using Postprocessor**).

Interaction Overview

The following Sequence Diagram illustrates what happens when a user invokes a service protected with an interceptor:

Figure: Sequence Diagram of the E2E Bridge Security Interceptor



When a user sends a request to the E2E Bridge (1), the bridge invokes a preprocessor, passing it the input message, the Principal and the name of the operation to be called (2). The preprocessor returns the, optionally modified, input message and Principal or throws an exception if e.g. access to the service is denied (3). After the Principals role is verified against the required roles (4), the actual service is invoked (5). The business logic is executed and returns its result (6) which then is passed to a postprocessor (7). The returned output message, again optionally modified (8), is then passed back to the user (9).

Typical applications for a preprocessor are:

- Authentication of principal
- Applying roles to an already authenticated principal
- Decryption of parameters
- Logging

Typical applications for a postprocessor are:

- Encryption of parameters
- Removal of critical parameters
- Logging