# Security QoS Modeling (SQML)
***For***
# Enterprise DRE Systems (eDRE)

**CS 388: Model Integrated Computing, Fall 2006**
**PROJECT REPORT**

**Akshay V. Dabholkar**

***Abstract***

*The development of security-critical large-scale distributed software systems is a difficult and error-prone process. Managing security manually and correctly with a sufficient level of assurance is tedious. Security is a tightly integrated and wide-spanning aspect of any system. So it's the responsibility of both the system developers as well as administrators. Each has a distinct role to play during the system lifecycle to provide security assurances. This has led to the approach of integrating security engineering and model-driven engineering concepts in a unified way. Since a large number of distributed, real-time and embedded enterprise systems are component based, the approach described in this paper focuses on applying security to CORBA based component technologies like the CORBA Component Model (CCM).*

## Introduction

Today, the research directions are more towards making large-scale, complex software systems *Trustworthy*. Enterprise and Distributed Real-time and Embedded are one such kind of systems. However there has been a lack of substantial research efforts towards providing guarantees and assurances regarding the non-functional aspects of such system like Quality of Service, adaptability, assurance and security. Enterprise-level and Distribution are two key aspects of these kinds of systems that play a significant role in making problem-solving difficult. These two aspects mean a large base of end-users of the system that can potentially access a large pool of available resources. There is a need to provide access control to prevent unauthorized access from these users. At the same time applying access control on a large scale is a very error-prone and tedious task. Security is mainly implemented at the platform level through role configuration but this is not sufficient. More complex security policies should be enforced as part of the implementation of the business code. This is obviously a large obstacle to one of the main principles of component based software development, namely component reusability. So in order to simplify declaration of security concerns as well as ensure their correctness and enforcement, a model-driven approach is necessary.

Interestingly, it has been observed that security assurances also form a part of the Quality of Service aspect of such systems. Security is one of the key QoS Aspects in addition to Real-Time, & Fault Tolerance in making Enterprise DRE (eDRE) systems *Trustworthy*. Security is just one of the dimensions of trustworthiness. Other dimensions are Fault Tolerance and Real-Time. Apply the MDE approach to security to be able to express security policy annotations at a higher level and evaluate conflicting requirements at modeling time. Goal is to add model-driven security enhancements to component middleware like CCM. Advanced platforms like CCM also offer support for the nonfunctional aspects like adaptability, flexibility, robustness and security and the functional and non-functional aspects are clearly separated from each other. The business functionality is implemented in a component whereas non-functional aspects like access control rules are handled by the container, the component's runtime environment and are described through policies defined using policy definition languages.

## Challenges

1) **Scalability:** Any tightly integrated approach like security is always difficult to scale when the number of components in the system increase/decrease dynamically at runtime and security policies need to be enforced/propagated to them.

2) **Granularity:** Being integrated and being able to scale means to be applicable at different levels of granularity in order to do so.

3) **Correctness:** In the light of application of security it is important to ensure that the other system functionalities work well in tandem with security functions to give a correct and secure system.

4) **Understandability:** Instead of handwriting and manually coding the security policies and enforcing them, they need to be able to be declaratively specified in the system deployment plan or at development time so that the necessary runtime hooks are initialized and code is generated.

5) **Performance:** Being integrated involves substantial checks and enforcement by the container and runtime which will incur significant performance penalties. So there is a need to incorporate dynamism in the security policies so that security can be swapped out of the system or diluted whenever needed.

## Types of protection

Protection is defined as "Freedom from worries". This definition of protection is more general contrary to the definition of security that is just defined in terms of authentication and authorization by most sections in the enterprise domain.

1) **Authorization:** Access control and data protection (untrusted environment). Critical data and resources need to be protected from unauthorized manipulation and access.

2) **Accountability:** Audit (weaker) and non-repudiation (stronger). There is a need to record the activities of users in order to be able to track their usage patterns and generate irrefutable evidence of their actions in the system. This enables administrators to catch infiltrators and access violations. These help quantify the access violations that pass undetected when hackers bypass the system firewall. So detection of violations is still possible in later stages after a breakout.

3) **Availability:** Service continuity and disaster recovery. This kind of protection encapsulates fault tolerance and reliability non-functional aspects of the system.

4) **Assurance:** System safety and operational correctness. This encapsulates providing real-time guarantees about system performance and operation.

All these define the Quality of Protection (QoP) in an enterprise system. Our focus is to provide a Role-Based Access Control Mechanism (RBAC) and a policy definition framework which encapsulates various dimensions and levels of QoS. We describe in brief the CORBA security model which is leveraged by the CCM security model.

**CORBA Security Model Specification (v1.8)**

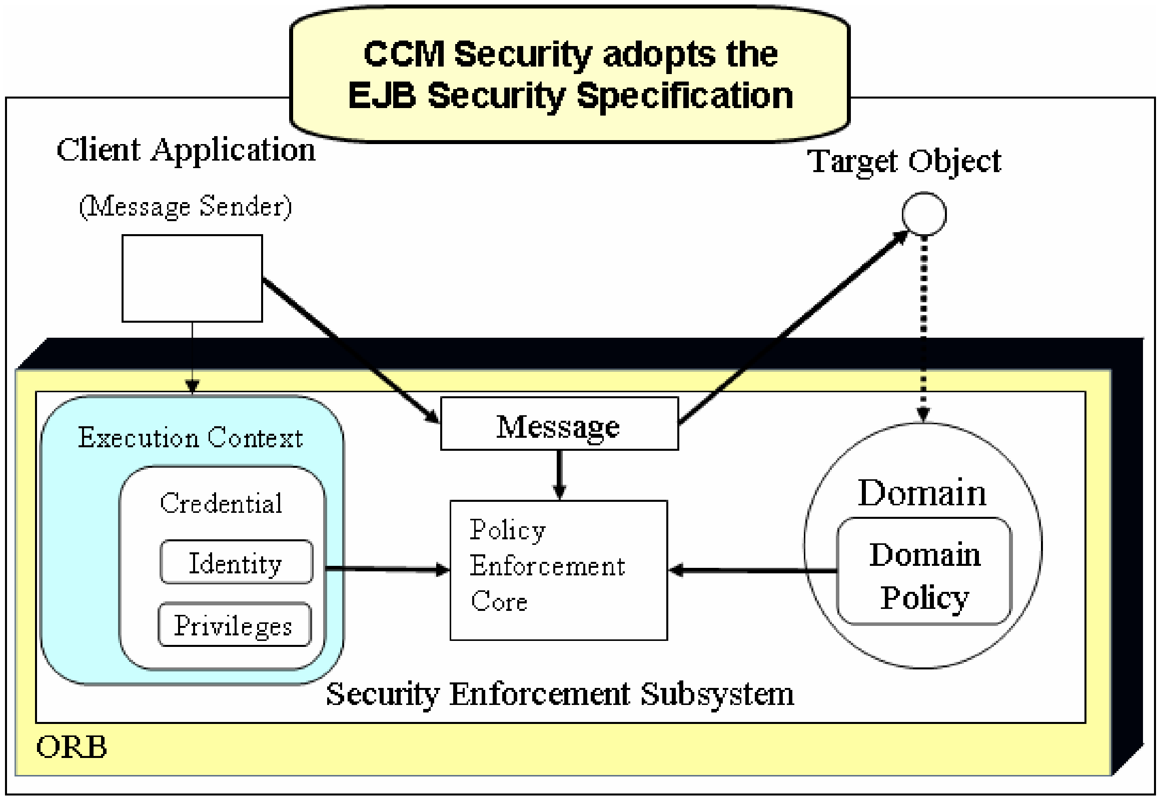


**Figure 1: CORBA Security model**

The block diagram shown above shows the overall structure of the CORBA security enforcement model. The CORBA security model defines protection based upon policy but it does not give the details of designing such policies. Policy may be domain specific but they need to be enforced by the ORB (Object Request Broker). The job of the ORB is to provide a PEP (Policy Enforcement Point) and PDP (Policy Decision Point). It also maintains a policy repository to store the policies used for runtime access violations and authorization checks.

The following block diagram summarizes the CORBA secure invocation model. In a component based systems components invoke operations on interfaces exported by other components in the system assembly.

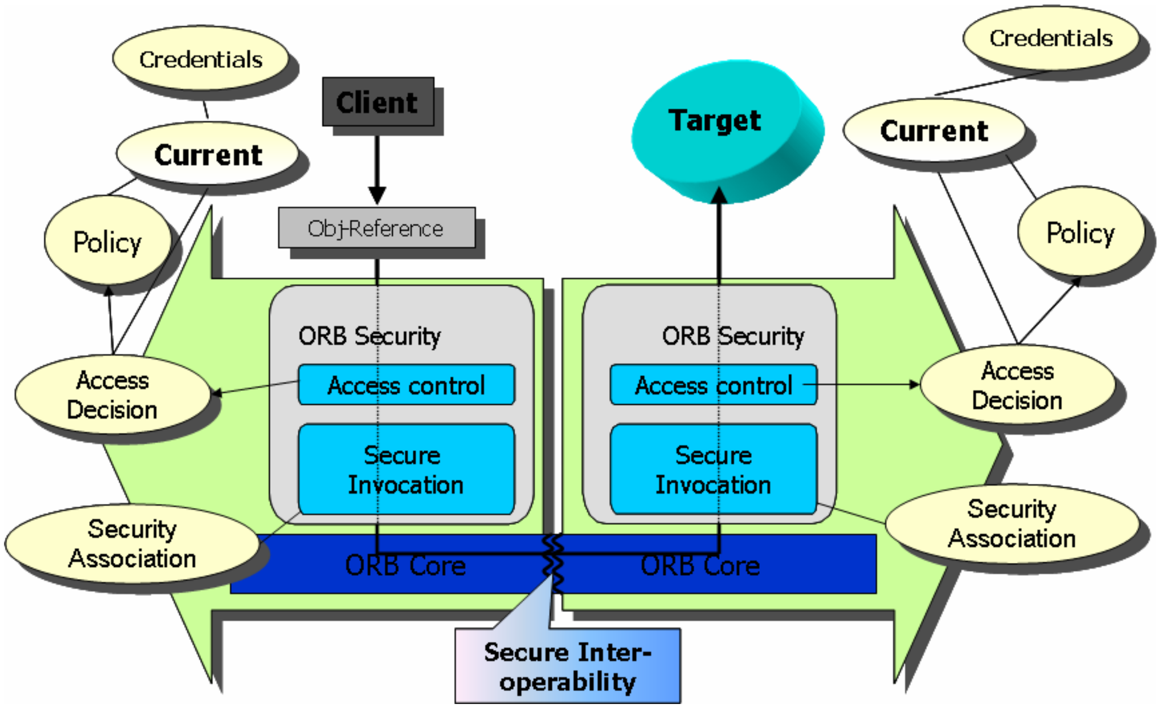


**Figure 2: CORBA Secure Invocation model**

The secure invocation in CORBA involves access control checks by the ORB based upon security policies which are used to take access decisions of whether the invocation is legitimate or not. These access control decisions and policies need to be checked at the client as well as server sides by the respective ORBs. They are centrally defined through security policies that are centrally located in a policy repository that is provided by an ORB service. The supporting mechanisms of authentication, integrity checks, and secure protocol implementations are leveraged by the ORB through ORB services that can be dynamically loaded when needed. Thus even with integrated security, the ORB's memory footprint is optimal.

## Security QoS Modeling

Access Control mechanisms enforce the security rules. They are used in environments that can be trusted to run a program to check whether the rules are violated. In environments like this, whenever anyone asks the system for access to a protected resource, the system checks its authorization rules to see whether the access is allowed; this is called an access control check. If the rules permit the access, the system proceeds normally, but if the rules forbid the access the system generates an error message and doesn't allow the access. Other kinds of protection as mentioned before are specified as QoS parameters to configure the ORB which implements and enforces them.

Components are composed by interconnecting their compatible interfaces to form an application assembly. A CCM application is composed of an assembly of components as shown below.

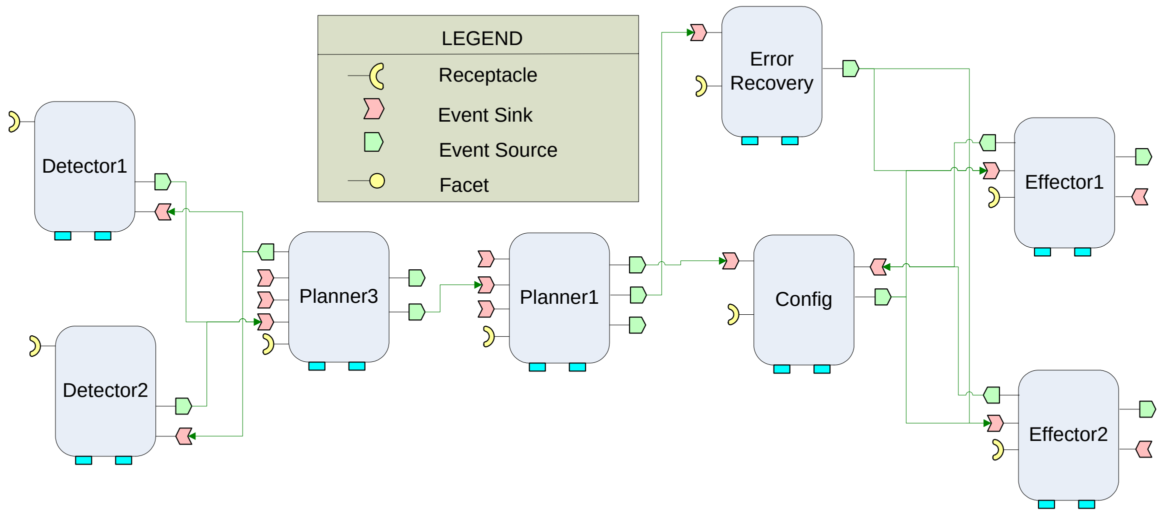


**Figure 3: An example CCM application assembly**

The Secure QoS Modeling Language (SQML) leverages and extends the Component QoS Modeling Language (CQML) to provide 3 levels of QoS enforcements: At component interfaces, the component itself and the entire component assembly. This enables the ability to express security concerns and QoS parameters at different levels of an application assembly.

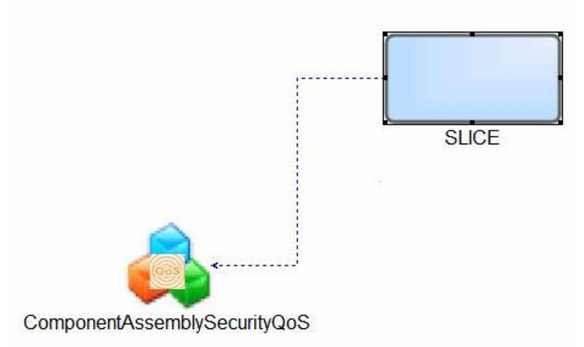
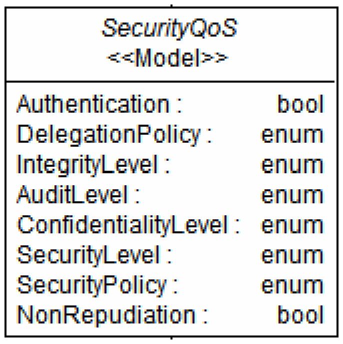


**Figure 4: Assigning Assembly Security QoS**

Each QoS model can be used to set different security parameters as shown below which enables to configure the system for different security requirements like authentication, security levels, audit levels, integrity level, kinds of credential delegation, policies enforced, etc.



*SecurityQoS*
<<Model>>

| | |
|---|---|
| Authentication : | bool |
| DelegationPolicy : | enum |
| IntegrityLevel : | enum |
| AuditLevel : | enum |
| ConfidentialityLevel : | enum |
| SecurityLevel : | enum |
| SecurityPolicy : | enum |
| NonRepudiation : | bool |

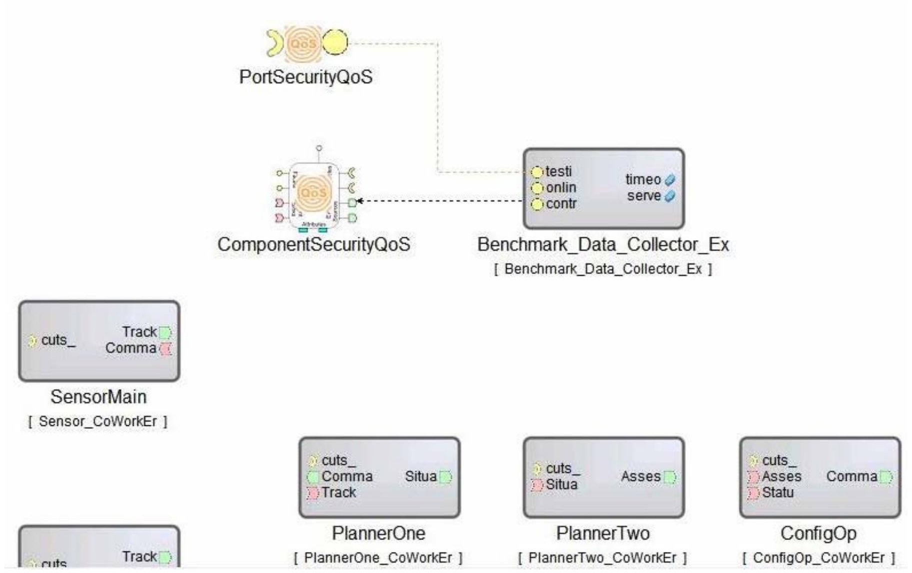


**Figure 5: Assigning Port and Component Security QoS**

## Access Control Granularity

SQML allows for fine-grained as well as coarse-grained access control granularity.

*Fine-grained:*
- Interface Operation
- Assembly Property
- Component Attribute

*Coarse-grained:*
- Interface
- Set of Operations
- Class of Operations (based on Required Rights - corba:gsum)
- Inter-Component Execution Flow (Path in an Assembly)

CORBA defines the concept of Rights which enables operation classification. The main concept is that a component developer is usually the only person who knows what kind of semantics each of his operations have. CORBA defines 4 types of rights for each method operation: getter, setter, use and management. These rights are assigned to operations as well as to user roles. These roles can be platform (CCM) specific as well as application specific. CORBA also allows the implementers to define their own set of access rights for their own semantics but they are enforced according to the matching (any/all) mechanism defined by the CORBA specification.

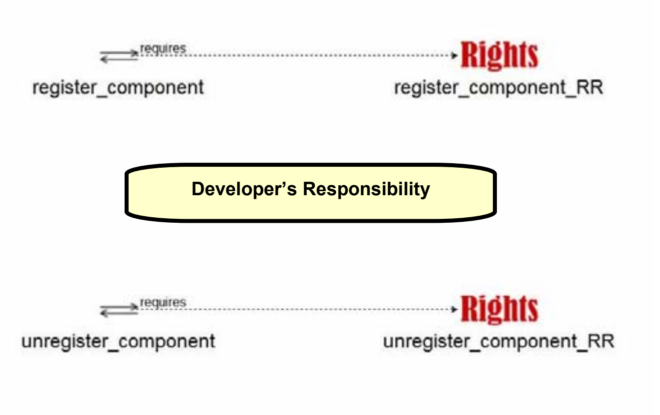


**Figure 6: Rights → Operation Classification**

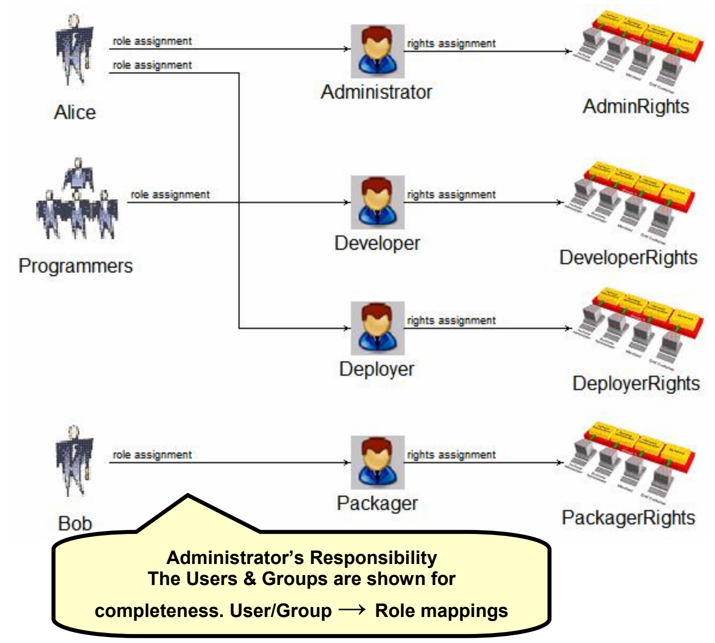


**Figure 7: User/Group → Role → Rights Mapping**

The user rights in their respective roles are matched against the method's rights to determine whether the method can be invoked. The rules define whether to allow the access or not even if the rights match but the rights have to match in order to define the rule. Thus the rules guarantees model level checking of rights which reduces the burden from the ORB as it just has to match the rules against method accesses by the user roles.

## Policy Definition Rules

In accordance with the different types/levels of Security QoS, there are three different kinds of rules for each kind: PortRule, ComponentRule, and AssemblyRule. All the three QoS types can have a port rule. An AssemblyRule has special semantics where security can be enforced on a critical path in the assembly. For example, a critical path in an application assembly can be a chain of components, their ports, respective methods that get accessed when the user invokes a business method on the initiating component. This path can represent the part of a workflow or a critical functionality like creating a new account in the banking application.

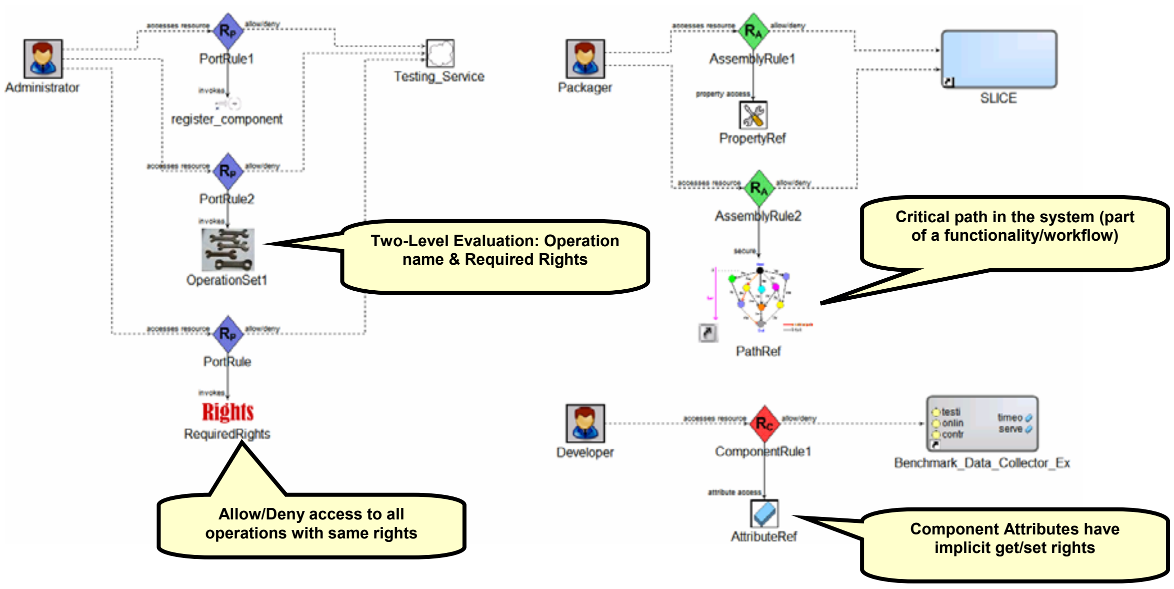


**Figure 8: Rule Construction**

## Interpreter Design

The SQML interpreter has been designed using the Model-View-Controller (MVC) design pattern The interpreter uses sophisticated template containers to capture the different Security QoS requirements which constitute the model (M). The Visitors that capture these requirements form the part of the controller (C) whereas the dumpers form the part of the view (V) that dump out XML descriptors for the model. This structuring is very important from the perspective of the deployment framework and also simplifies extensibility and enables flexibility. The SQML interpreter compresses a lot of repetitive model parsing and visitor code through powerful template metaprogramming and also automates the management of the interpreter XML Document stack through auto_ptr type of implementation.

The interpreter has five different sub-modules:
1) The operation-rights classification interpreter
2) Role-Rights mapping interpreter
3) Security Policy interpreter
4) Security QoS assignments interpreter
5) Permissions generator

The interpreter can be invoked by pressing the “ ” button. Some of the tasks of the interpreter are summarized as follows:

- *Inject security related assertions and constraints to the CCM component deployment and configuration plan and generate security policies and permissions*
- Generate User → Role → Granted-Rights mappings defined by the system administrator for deployment in Security Assertion Markup Language (SAML) like syntax.
- Generate Operation → Required-Rights mappings for an interface that are determined by the component designer. If rights are defined on the interface then they are inherited by its operation for whom any rights have not been defined.
- Generate Policy definition files in eXtended Access Control Markup Language (XACML) like syntax.
- Based on the mappings and policy rules, it will generate method permissions that are defined on the operations of the interface definition for the User roles that have rights to invoke a method
- Generate additional metadata to configure the CCM container for applying the defined Security policies and enforcing them on the object interactions

## Benefit of Security Modeling

**1)** SQML allows for expressing of cross-cutting concerns that can be implemented at the interface level, component level as well as component assembly level.
**2)** It enables conflict resolution between policy rules in a general way.
**3)** It enables well-defined policy definitions.
**4)** It sheds off some responsibility from the ORB through definition of well-formed policies, rule combining and conflict resolution.
**5)** Provides a higher level tool for declarative security specification for the deployment of large-scale component-based systems.
**6)** It incorporates security into the QoS aspects of component systems which is an important step towards complete QoS modeling of such systems enabling their trustworthiness.

## Future Work

- The language allows modeling of security QoS with much more generality and flexibility than existing solutions (OpenPMF)
- The interpreter generates a permissions file which can be used by deployment tools to integrate security properties with application deployment

  Some benefits which can be extracted from the language (but not implemented now) are listed below.

- Extend the critical path functionality to provide business workflow and process based access control security spanning an entire application
- Define efficient rule and policy validation and rule combining algorithms.
- Provide middleware infrastructure support for security in the CCM container through container portable interceptors, leveraging the facilities of the CORBA security service implementation available with TAO (The ACE ORB).

## References

[1] The CORBA Security Service Specification v1.8, OMG document formal/02-03-11

[2] Object Management Group. Model Driven Architecture Web Page. www.omg.org/mda. Needham, MA, May 2003

[3] ObjectSecurity. OpenPMF Project. http://www.openpmf.org

[4] Lang, U., Schreiner, R. OpenPMF Security Policy Framework for Distributed Systems. Proceedings of the Information Security Solutions Europe (ISSE 2004) Conference, Berlin, Germany, September 2004

[5] QoS for CCM, OMG document, ptc/06-04-15

[6] CORBA Component Model Specification 4.0, OMG document formal/06-04-01

[8] UML Profile for QoS and Fault-Tollerance Specification, OMG document formal/06-05-02

[9] CORBA Specification, OMG document: formal/04-03-01