Building Secure Architectures Using Security Patterns

Eduardo B. Fernandez

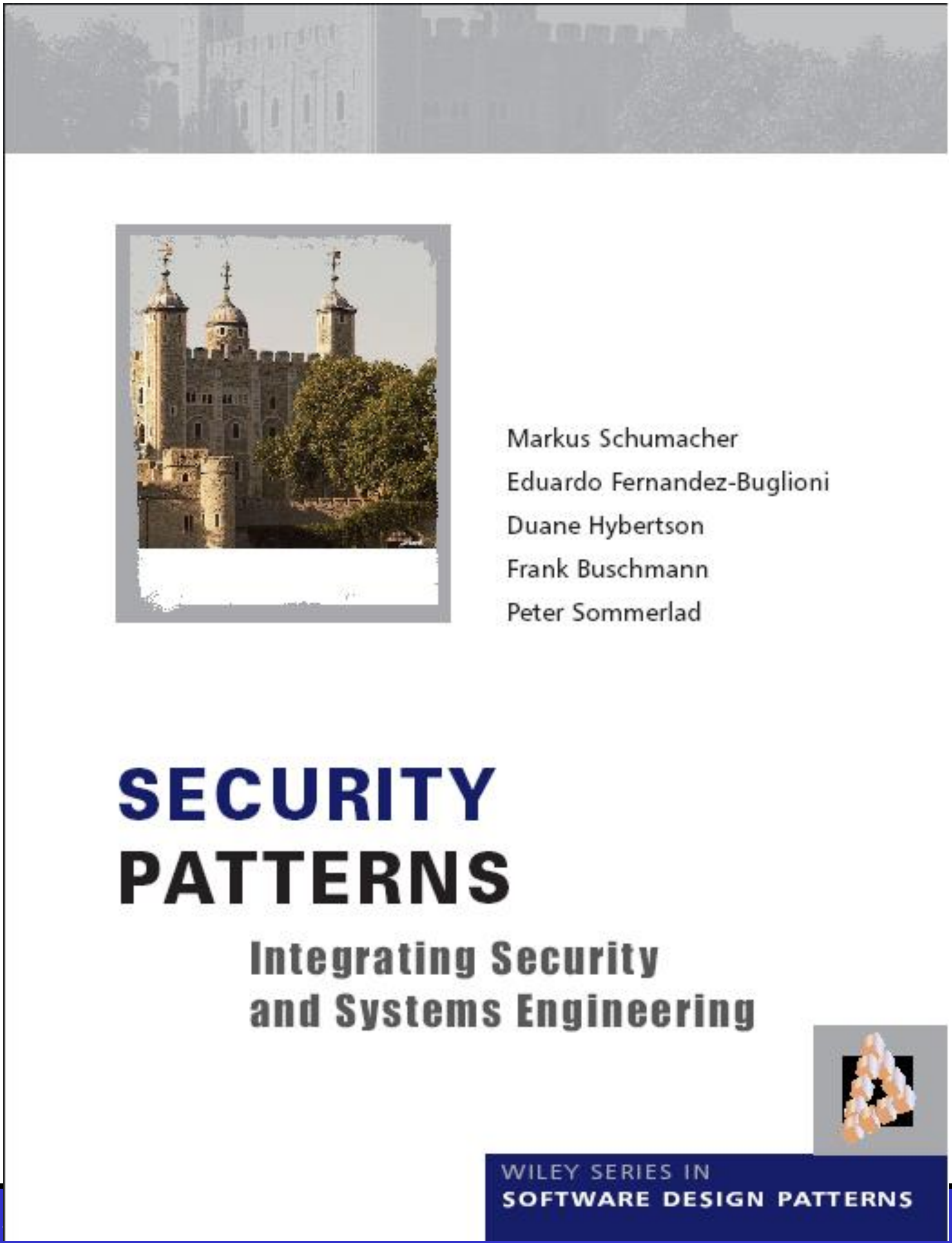
Dept. of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL, USA

http://www.cse.fau.edu/~ed

ed@cse.fau.edu

About me

- ***Professor of Computer Science at Florida Atlantic University, Boca Raton, FL., USA***
- ***At IBM for 8 years (L.A. Scientific Center).***
- ***Wrote the first book on database security (Addison-Wesley, 1981).***
- ***Author of many research papers***
- ***Consultant to IBM, Siemens, Lucent,…***
- ***MS EE Purdue U, PhD CS UCLA***

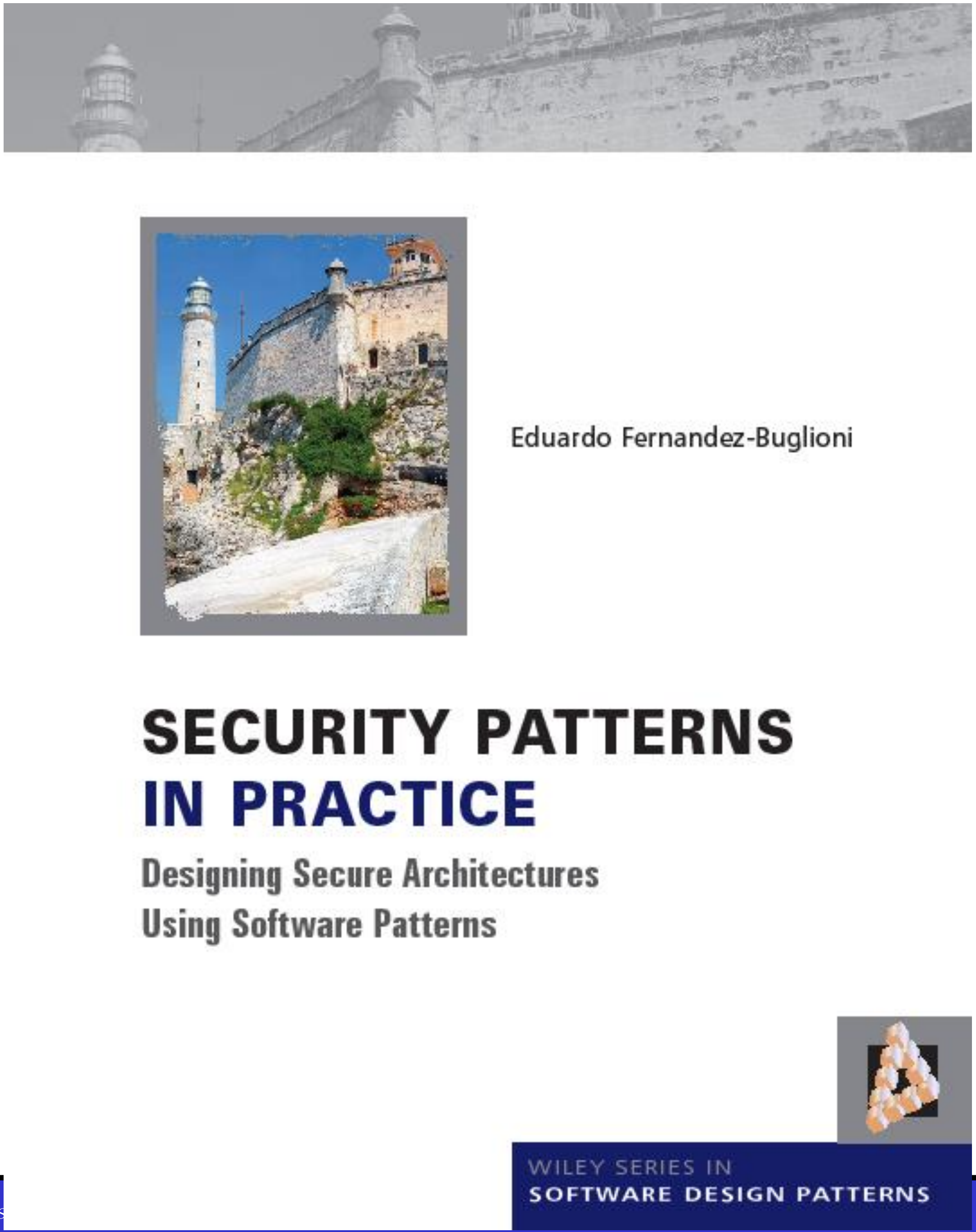


Markus Schumacher
Eduardo Fernandez-Buglioni
Duane Hybertson
Frank Buschmann
Peter Sommerlad

SECURITY PATTERNS

Integrating Security and Systems Engineering

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS



Eduardo Fernandez-Buglioni

SECURITY PATTERNS IN PRACTICE

Designing Secure Architectures Using Software Patterns

WILEY SERIES IN SOFTWARE DESIGN PATTERNS

Objectives

- *Get a panorama of security patterns and their use*
- *Study some specific patterns in detail*
- *Consider a systematic approach to build secure systems based on patterns and UML*
- *Use of reference architectures*

Outline

- ***Security concepts***
- ***Attacks/threats***
- ***The design of secure systems: relating threats to use cases.***
- ***Security patterns***
- ***Security models and their patterns---policies, access matrix, multilevel models, RBAC***
- ***Authentication. Authorization, RBAC and security policies.***
- ***A secure design methodology***
- ***Security reference architectures (SRAs), a SRA for cloud systems***
- ***Conclusions***

The value of information

- ***Individuals and enterprises rely on information for credit, health, professional work, business, education,…***
- ***Illegal access (reading or modification) to information can produce serious problems***
- ***Because of its value, information is a growing target of attacks***

Security is the protection against:

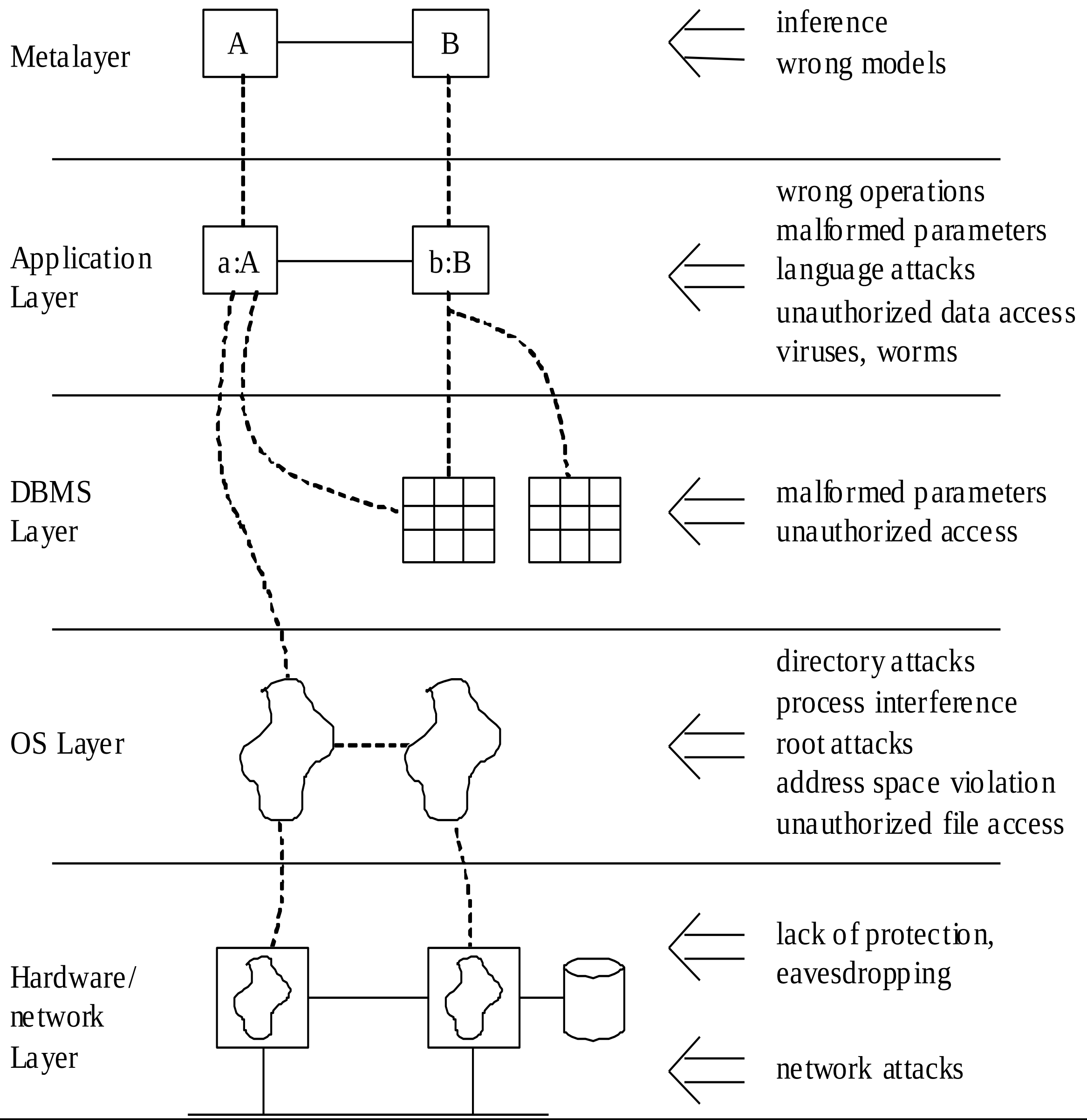
- *Unauthorized data disclosure* ***(confidentiality or secrecy).***
- *Unauthorized data modification* ***(integrity). Unauthorized modification of data may result in inconsistencies or erroneous data. Data destruction may bring all kinds of losses.***
- *Loss of availability (Denial of service)****—Users or other systems may prevent the legitimate users from using their system.***
- *Lack of accountability****—Users should be responsible for their actions and should not be able to deny what they have done (non-repudiation).***

Countermeasures (defenses)

- ***Identification and Authentication*– first step**
- ***Access control/ authorization* --provide confidentiality and integrity**
- ***Information hiding*– making information unintelligible**
- ***Auditing-- basis for prosecution or improvements* to the system**
- ***Intrusion detection*—attack alerting**

Threats

- ***We need to understand the threats to the system to decide how to defend it***
- ***Excess of security mechanisms results in loss of performance, extra complexity, and higher costs***
- ***The objective is to provide an appropriate defense according to the value of our assets***


Metalayer
A
B
inference
wrong models
Application Layer
a:A
b:B
wrong operations
malformed parameters
language attacks
unauthorized data access
viruses, worms
DBMS Layer
malformed parameters
unauthorized access
OS Layer
directory attacks
process interference
root attacks
address space violation
unauthorized file access
Hardware/ network Layer
lack of protection, eavesdropping
network attacks

Current situation

- ***The Internet is an insecure place and attacks keep occurring***
- ***One of the main reasons is the poor quality of the software used in systems and application software***
- ***Software engineering neglected security for a long time***

Possible remedies

- ***Help designers build secure systems using a systematic approach***
- ***Provide units of security (packed solutions to specific problems)***
- ***Build security together with the functional part of the application***
- ***We use a model-based approach***

Need for a conceptual approach I

- ***Security should be applied where the application semantics is understood***
- ***Security is an all-levels problem***
- ***We should start from high-level policies that can be mapped to the lower levels***
- ***We need precise models to guide system development***
- ***Consider a layered architecture as a guideline***

Need for conceptual structure II

- ***A unified system is easier to understand: better design, better administration***
- ***Easier to analyze effect of new hardware or software***
- ***Apply principles of good design***
- ***Start from policies and models***
- ***Apply security throughout the lifecycle***

Security principles

- *Holistic approach--Cover all architectural levels and all units*
- *Highest level—security constraints must be defined where their semantics are clear and propagated down*
- *Full mediation—every request for resources must be authorized*
- *Defense in depth—have more than one line of defense*
- *Closed system—Everything is forbidden unless explicitly authorized*
- *Least privilege—Assign only the necessary rights to perform functions*
- *Separate and compartmentalize*
- *Secure defaults*

Patterns

- ***A pattern is a solution to a recurrent problem in a specific context***
- ***Idea comes from architecture of buildings (C. Alexander)***
- ***Applied initially to software and then extended to other domains***
- ***Appeared in 1994 and are increasingly being accepted by industry***

Why security patterns?

- *Analysis patterns can be used to build conceptual models of software, design patterns can be used to make software more flexible and reusable, and security patterns can be used to build secure systems, including hardware or organizational problems.*
- *Security has had a long trajectory, starting from the early models of Lampson and Bell/LaPadula in the early 70s, and resulting in a variety of approaches to analyze security problems and to design security mechanisms. It is natural to try to codify this expertise in the form of patterns.*
- *Security patterns describe mechanisms to stop some attacks and apply principles of good design*

Value of security patterns

- ***Can apply security principles (Least priviliege) or describe security mechanisms able to stop specific threats(Firewalls)***
- ***Can guide the design and implementation of the security mechanism itself***
- ***Can guide the use of security mechanisms in an application (stop specific threats)***
- ***Can help understanding and use of complex standards (XACML, WiMax)***
- ***Convenient for teaching security principles and mechanisms***

POSA template

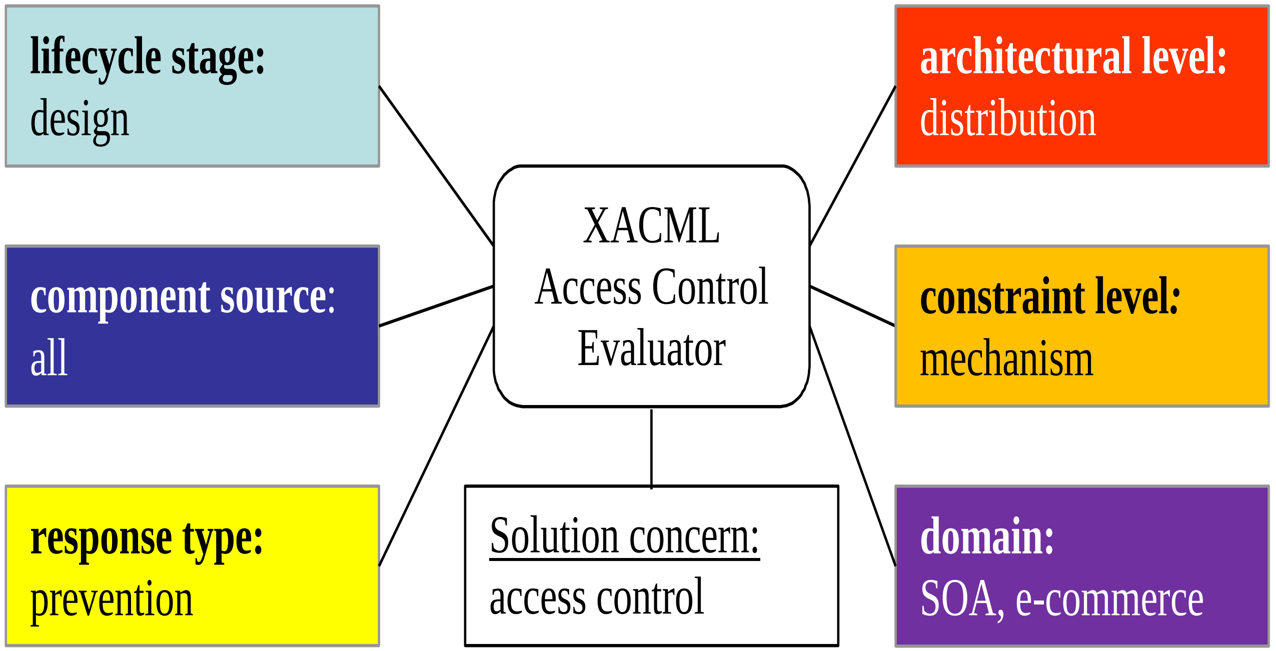
- ***Intent (thumbnail)***
- ***Example***
- ***Context***
- ***Problem and forces***
- ***Solution: in words, UML models (static and dynamic)***
- ***Implementation***
- ***Example resolved***
- ***Known uses***
- ***Consequences***
- ***See also (related patterns)***

Using the patterns

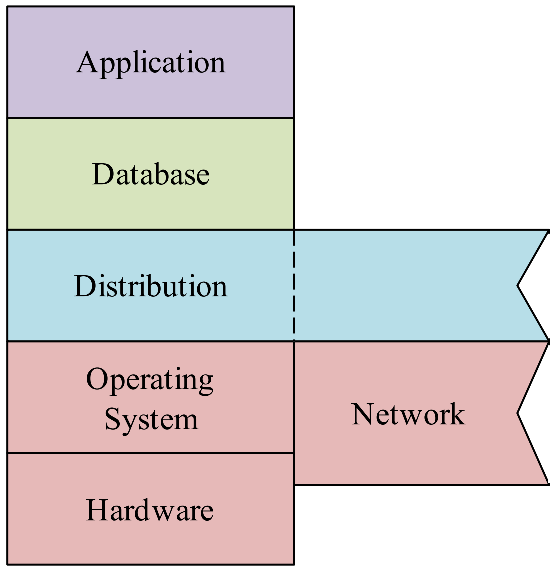
- ***Catalogs of patterns are not enough, designers must be given guidance in their use***
- ***There are many patterns (growing in number) and the task of selecting them gets harder***
- ***A first approach is to classify the patterns according to some criteria***

Sec Pattern classification

- ***Use a multidimensional matrix***
- ***Dimensions may include: architectural level, lifecycle stage, concern, type of pattern, domain,…***
- ***Example: XACML***


lifecycle stage:
design
component source:
all
response type:
prevention
XACML Access Control Evaluator
Solution concern:
access control
architectural level:
distribution
constraint level:
mechanism
domain:
SOA, e-commerce

Security layers



Authenticator

- ***How to verify that a subject is who it says it is? Use a single point of access to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject.***

Authenticator

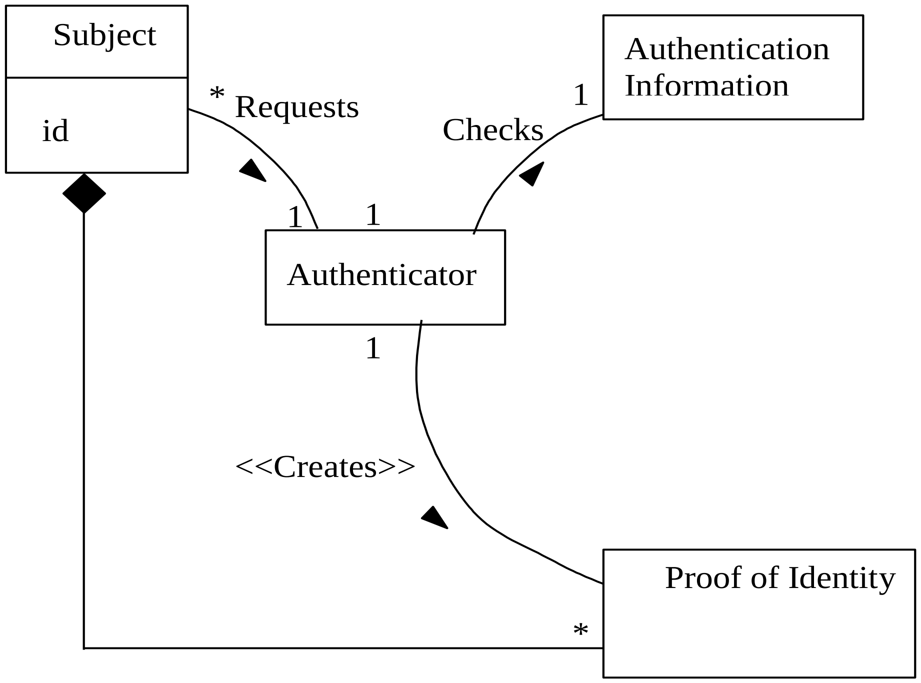


Figure 2 Authentication Pattern

Authentication

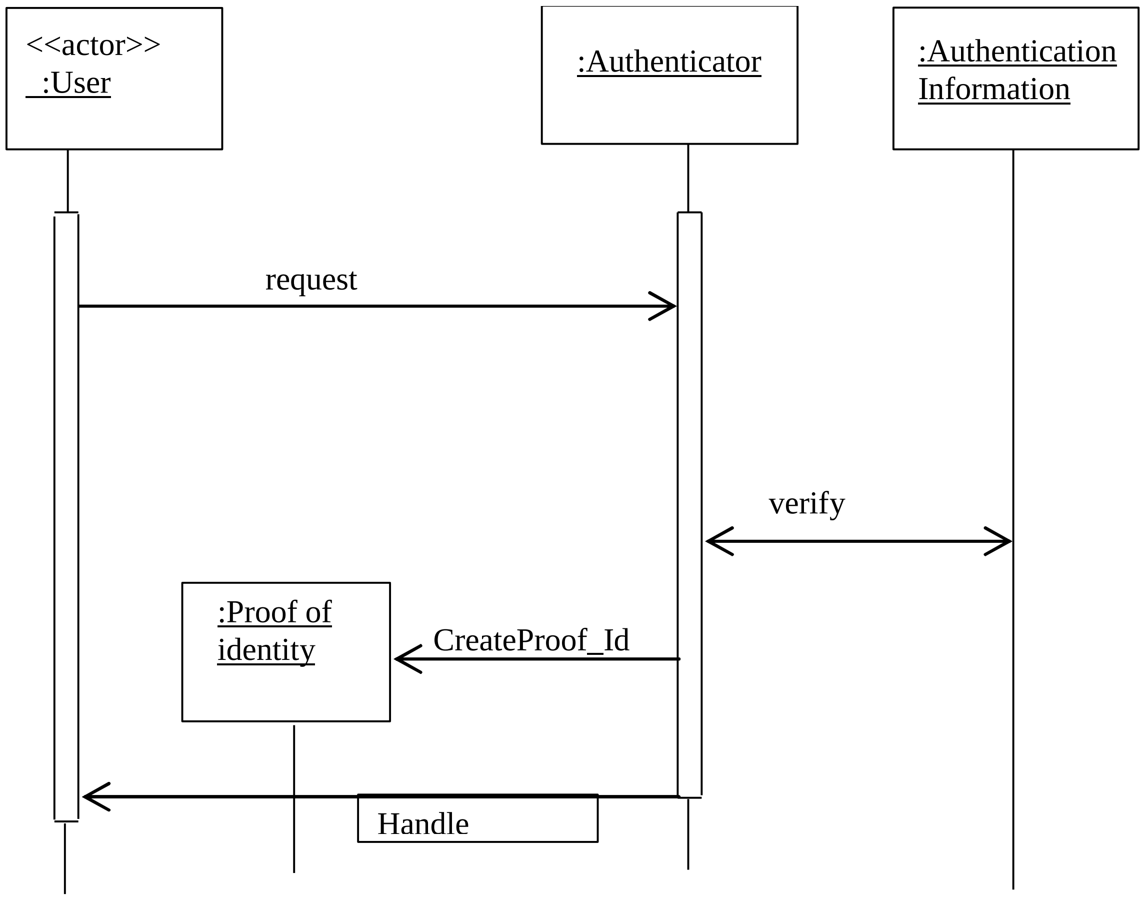
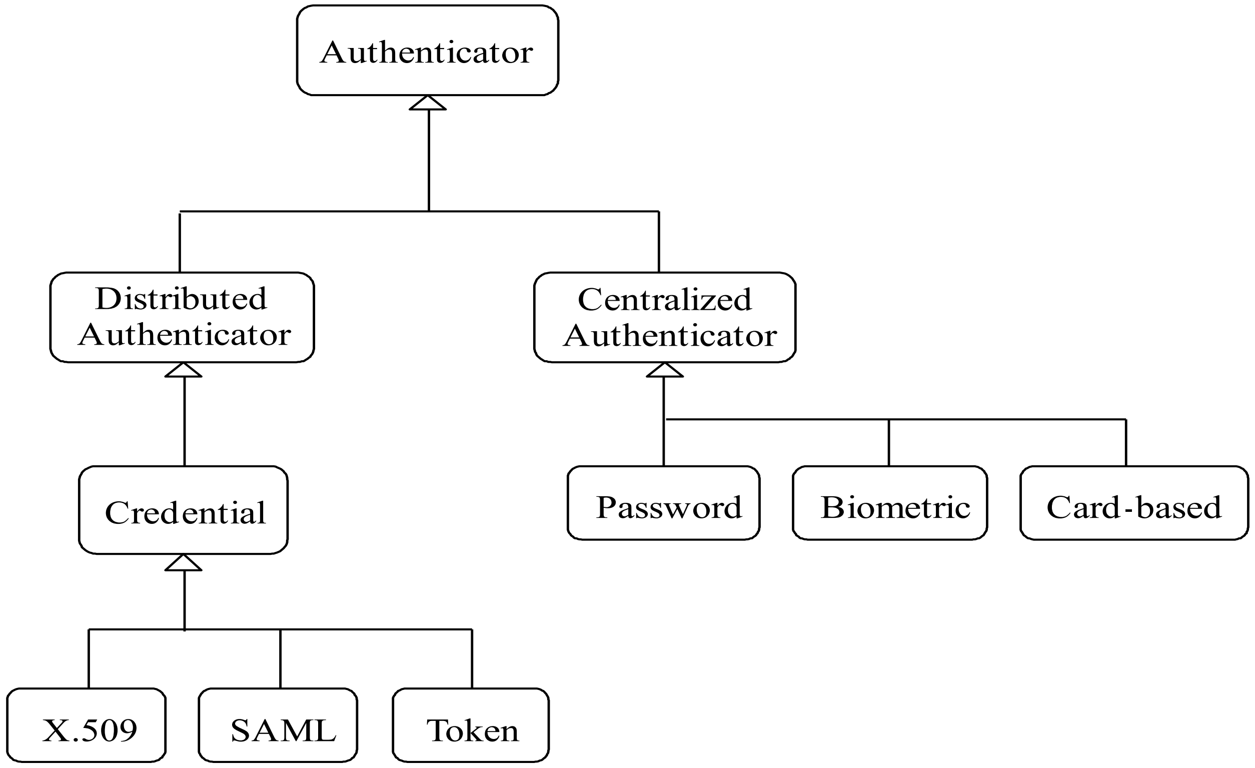


Figure 3 Authentication Dynamics

Authenticator hierarchy



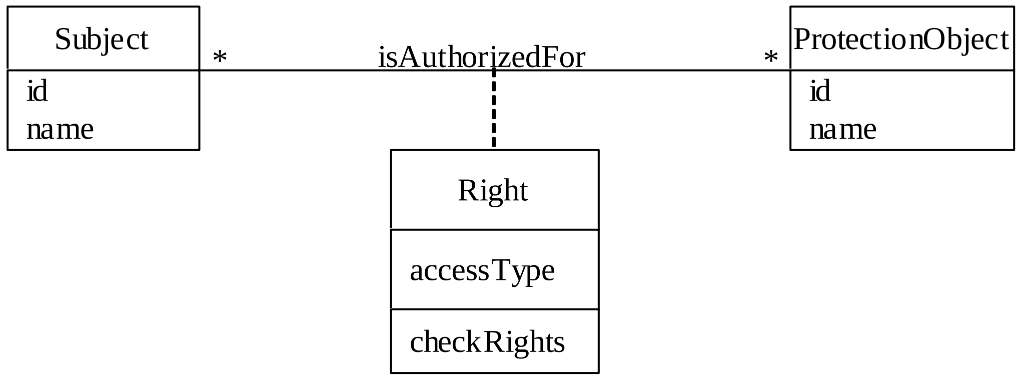
Access control

- ***Authorization:*** ***rules that specify who has access to what and how***
- ***Different models define rules appropriate for specific environments***
- ***Enforcement:*** ***intercept access requests and verify that they are authorized***
- ***Enforcement is embodied in a Reference Monitor***

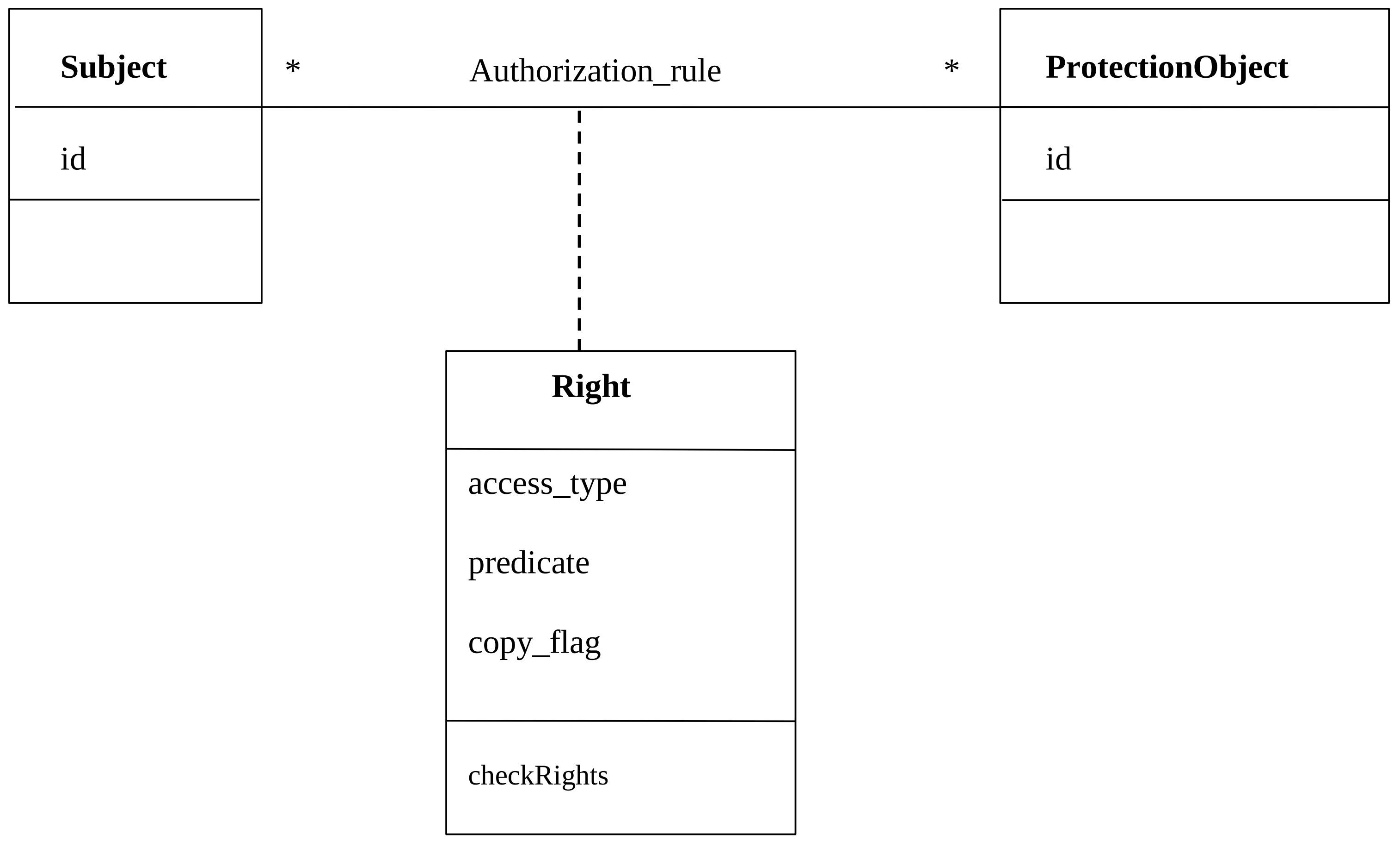
Applic. Layer: Access control models

- ***Authorization.*** ***How do we describe who is authorized to access specific resources in a system? A list of authorization rules describes who has access to what and how.***
- ***Role-Based Access Control (RBAC).*** ***How do we assign rights to people based on their functions or tasks? Assign people to roles and give rights to these roles so they can perform their tasks.***
- ***Multilevel Security. How to decide access in an environment with security classifications.***

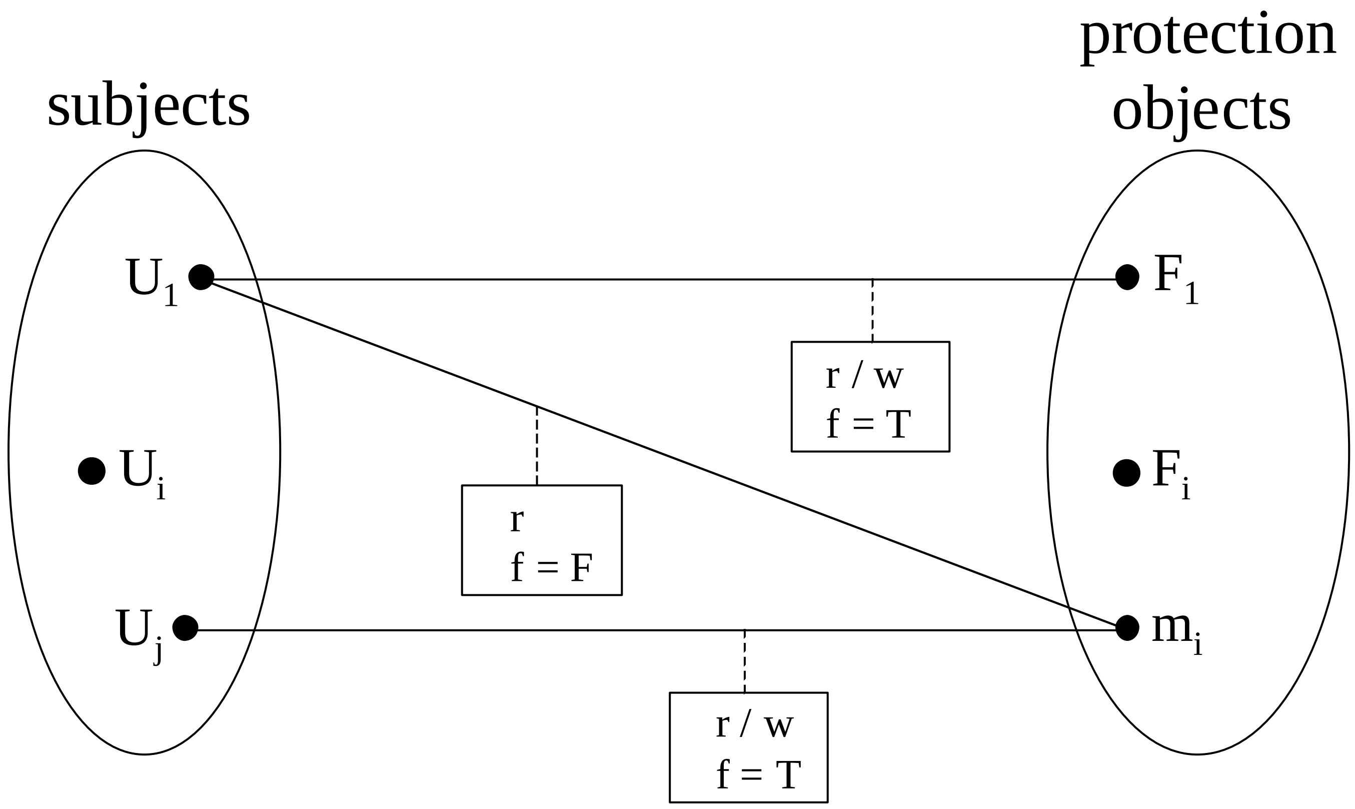
Authorization/Access Matrix



Access Matrix



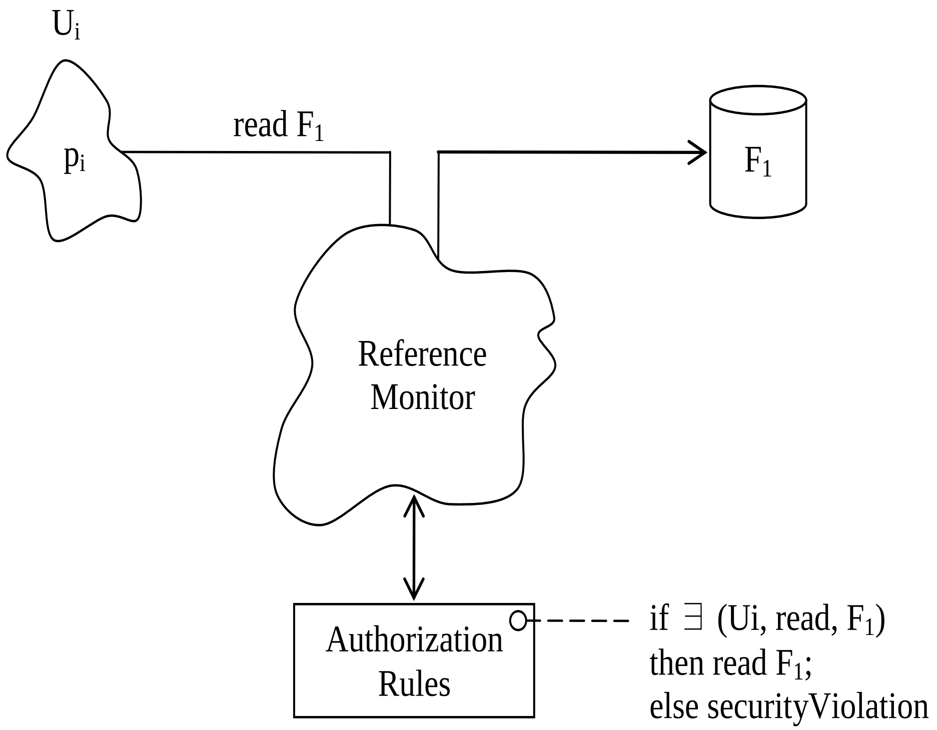
Authorization mapping



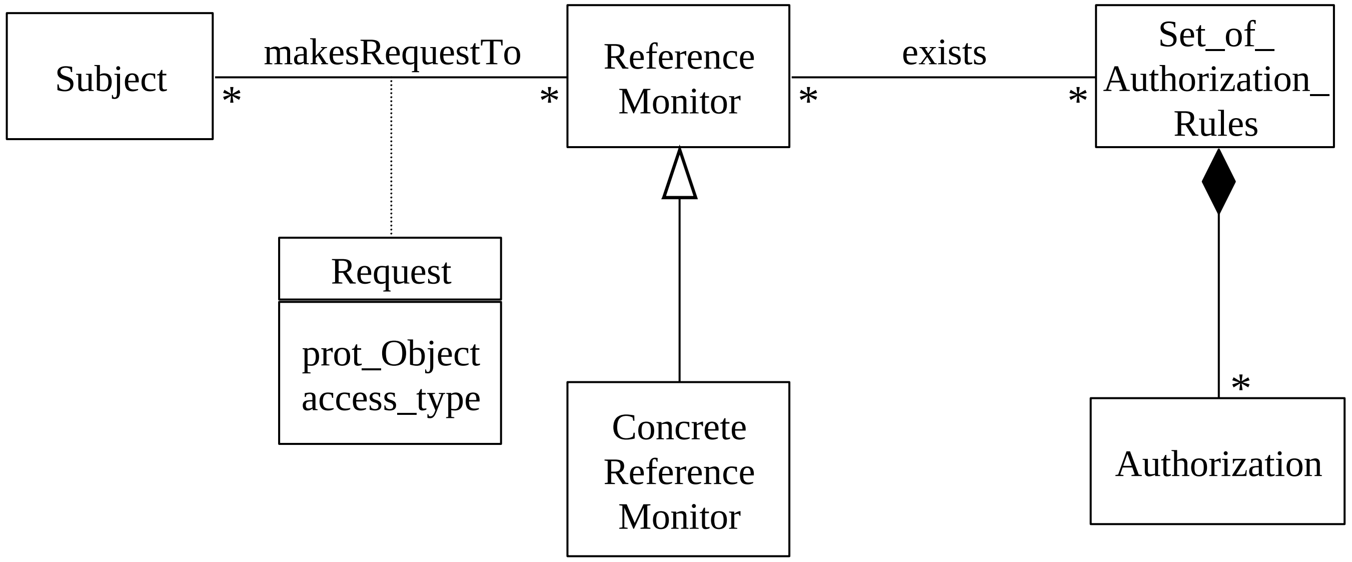
Reference Monitor

- ***Each request for resources must be intercepted and evaluated for authorized access***
- ***Abstract concept, implemented as memory access manager, file permission checks, CORBA adapters, etc.***

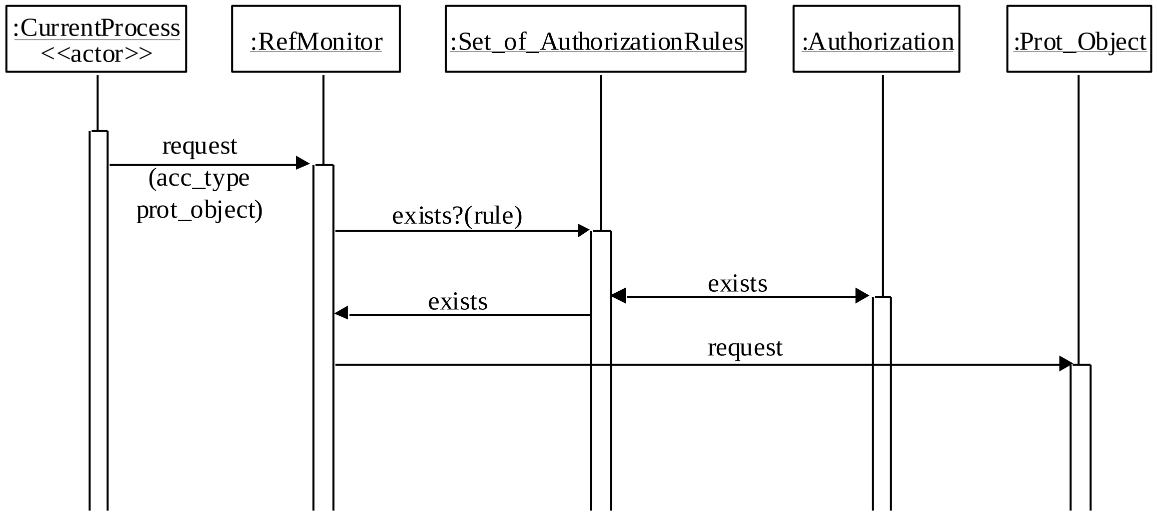
Reference Monitor idea



Reference monitor pattern



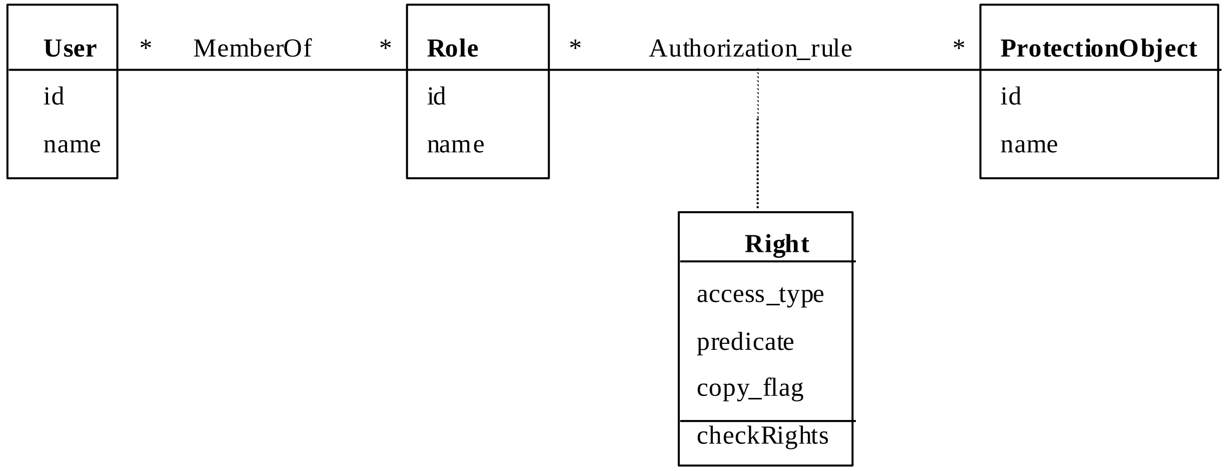
Enforcing access control



Role-Based Access Control

- ***Users are assigned roles according to their functions and given the needed rights (access types for specific objects)***
- ***When users are assigned by administrators, this is a mandatory model***
- ***Can implement least privilege and separation of duty policies***

Basic RBAC pattern

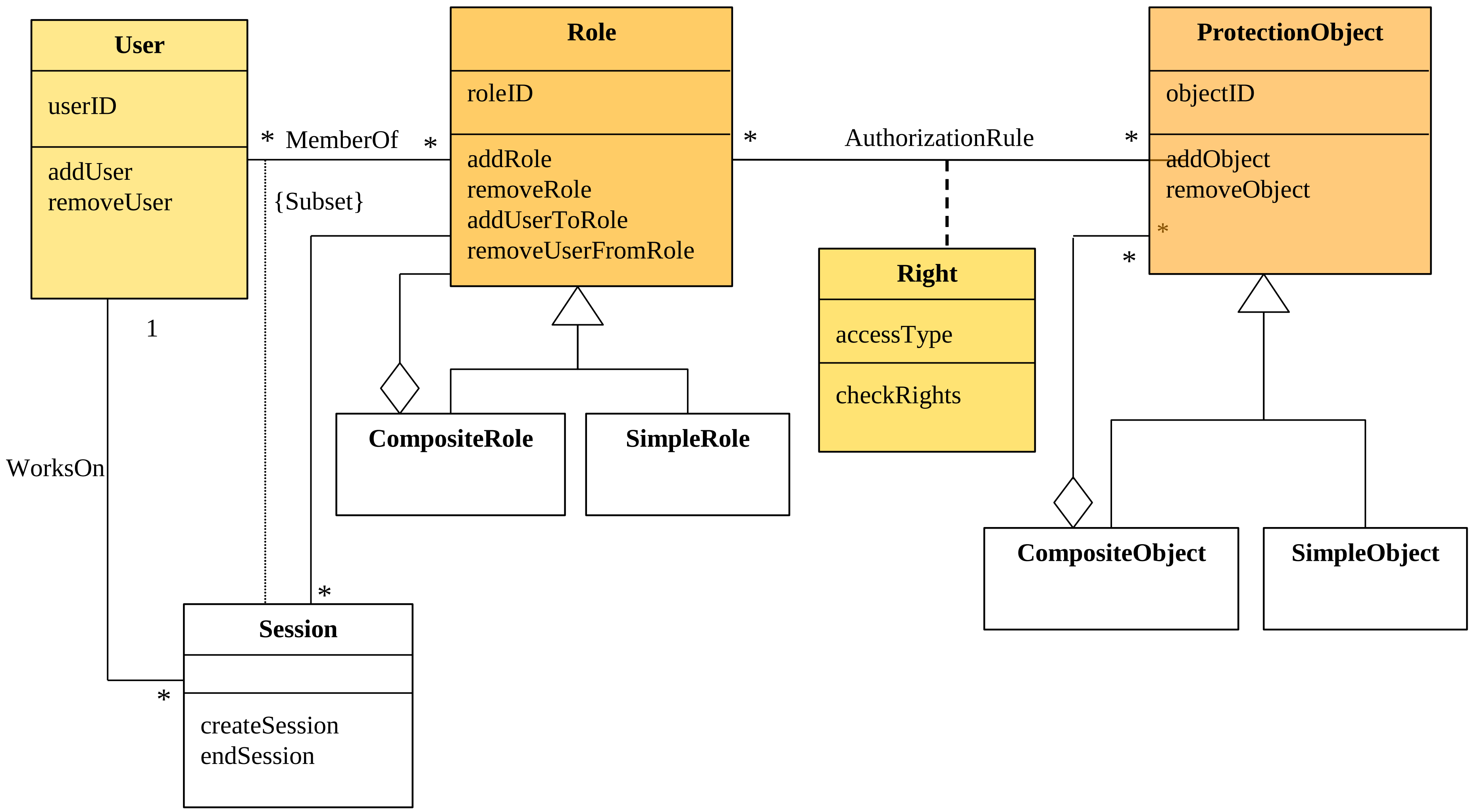


Growing models

- ***We can grow models by adding classes, e.g. from Authorization to RBAC we add a Role class.***
- ***We can add sessions as execution context***
- ***We can add hierarchies of roles and of objects (Composite pattern)***
- ***Subroles inherit role rights from superclass roles.***
- ***Access to an object gives implicit access to its subobjects***

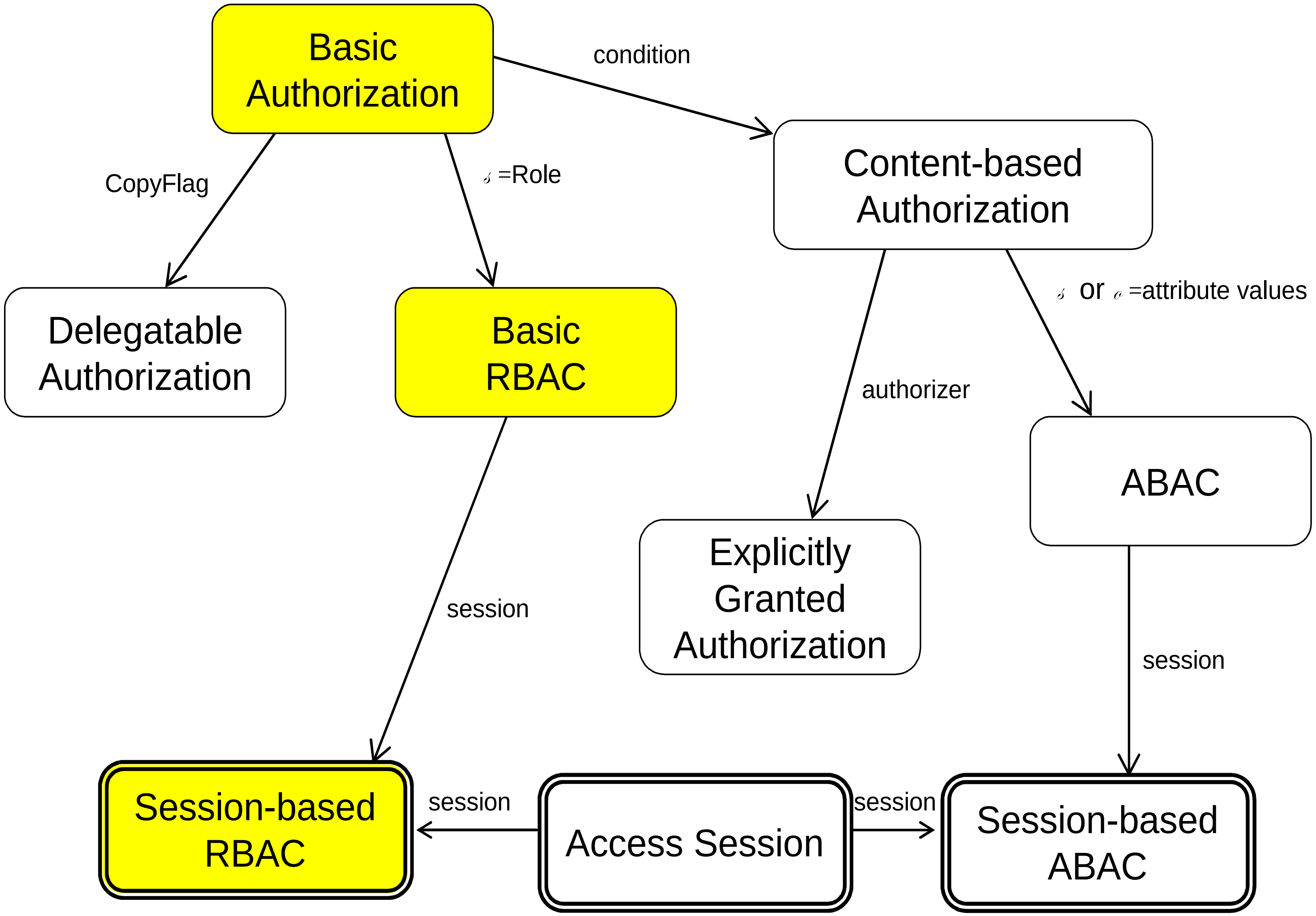
Extended RBAC

- ***Concept of session***
- ***Separation of administrative roles***
- ***Composite roles***
- ***Groups of users***


User
userID
addUser
removeUser
* MemberOf *
{Subset}
Role
roleID
addRole
removeRole
addUserToRole
removeUserFromRole
* AuthorizationRule *
ProtectionObject
objectID
addObject
removeObject
*
*
Right
accessType
checkRights
1
WorksOn
CompositeRole
SimpleRole
CompositeObject
SimpleObject
*
Session
createSession
endSession
*

Pattern diagrams

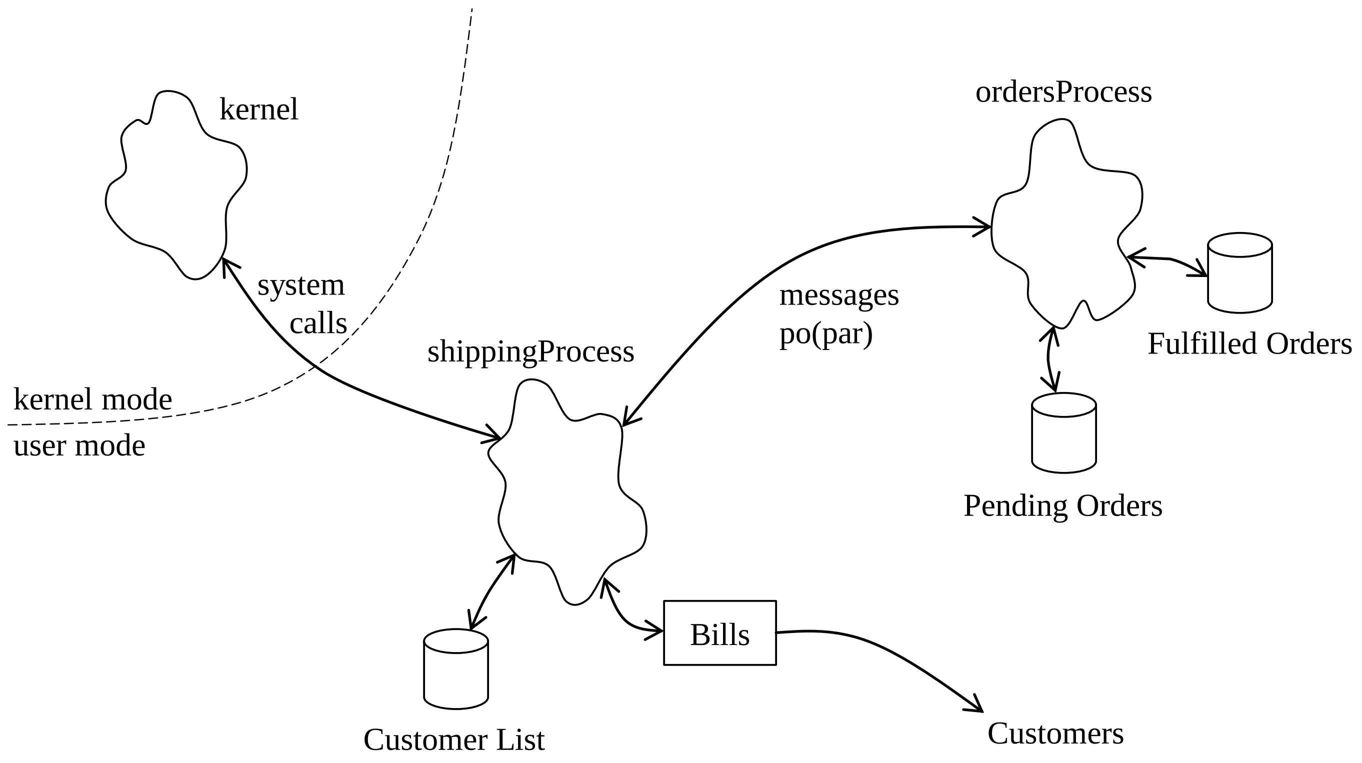
- ***Show relationships between patterns***
- ***Nodes are patterns***
- ***Links are relationships***
- ***Links are labeled with type of dependency or contribution***
- ***Useful to guide the designer in the selection of patterns***
- ***Can go in Introduction to a set of patterns, or in Context, or in See also***


Basic Authorization
condition
Content-based Authorization
CopyFlag
s =Role
Delegatable Authorization
Basic RBAC
s or o =attribute values
authorizer
ABAC
Explicitly Granted Authorization
session
session
Session-based RBAC
session
Access Session
session
Session-based ABAC

Operating system layer

- ***OS architectures***
- ***OS security features***
- ***Virtualization***

Process interaction

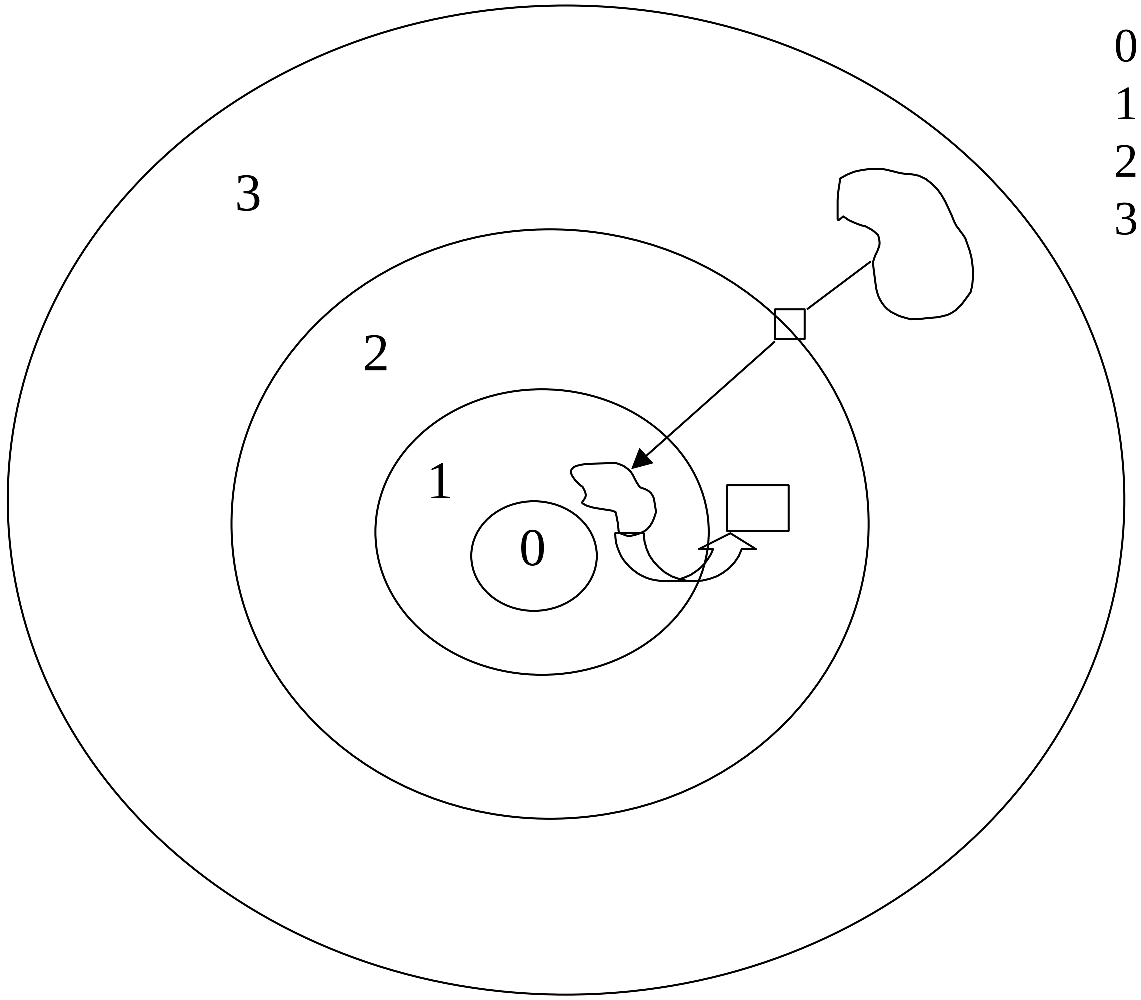


Process communication

- ***Process communication also has an effect on security. Systems that use explicit message passing have the possibility of checking each message to see if it complies with system policies.***
- ***A security feature that can be applied when calling another process is protected entry points. A process calling another process can only enter this process at pre-designed entry points. This prevents bypassing entry checks***
- ***The number and size of arguments in a gate crossing can also be controlled (this may protect against some types of buffer overflow attacks)***

Protection rings

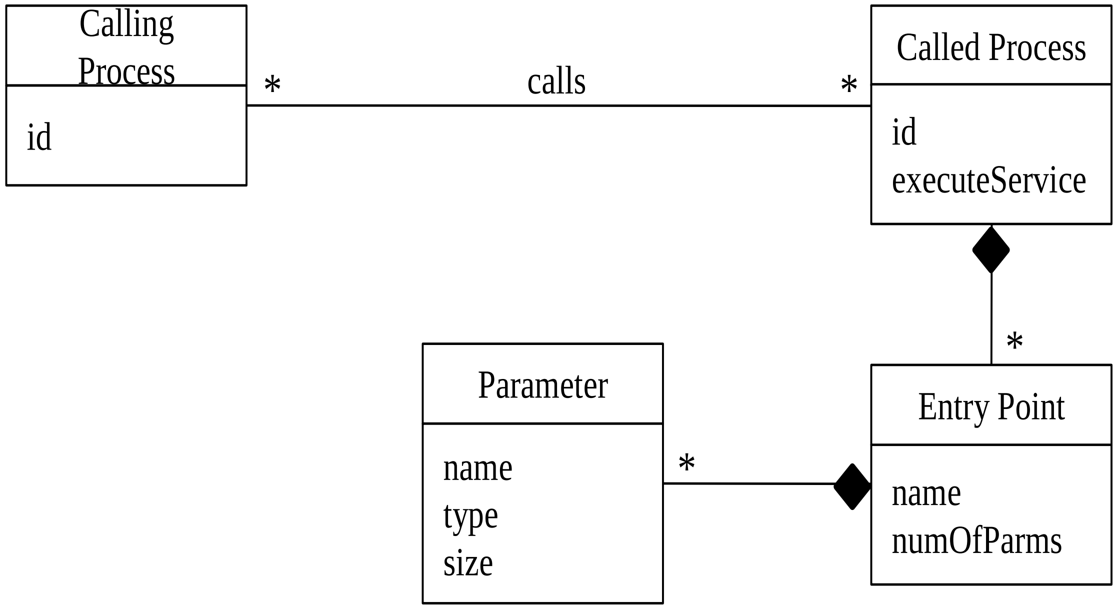
- ***Some architectures define in their hardware a set of rings (4 to 32) that correspond to domains of execution with hierarchical levels of trust. Rings are a generalization of the concept of mode of operation.***
- ***Crossing of rings is done through gates that check the rights of the crossing process. A process calling a segment in a higher ring must go through a gate.***



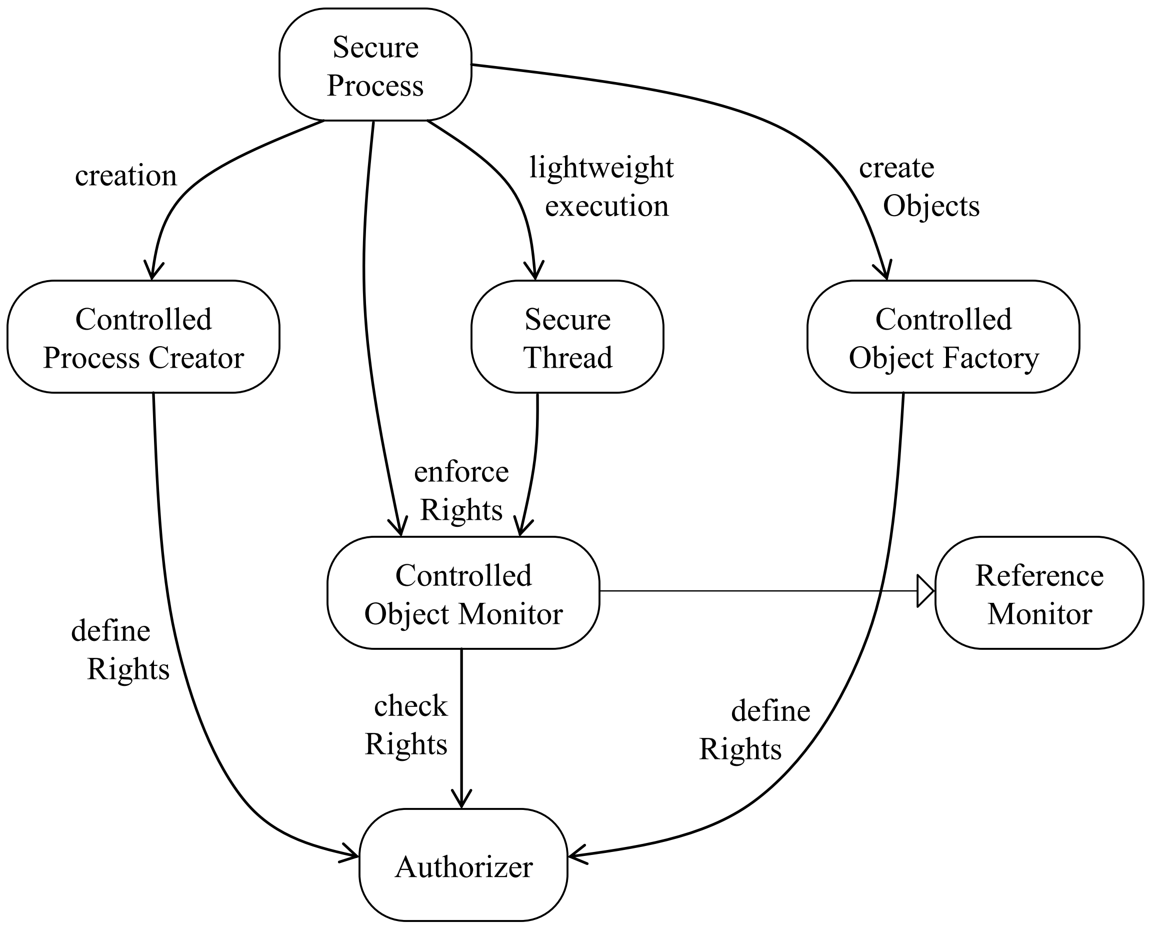
0 = kernel
1 = OS functions
2 = safe applications
3 = untrusted applications

- Calls upward
  (higher privilege)
- Data access toward
  less privilege
- Gate crossings
- Protected entry points

Protected Entry Point pattern



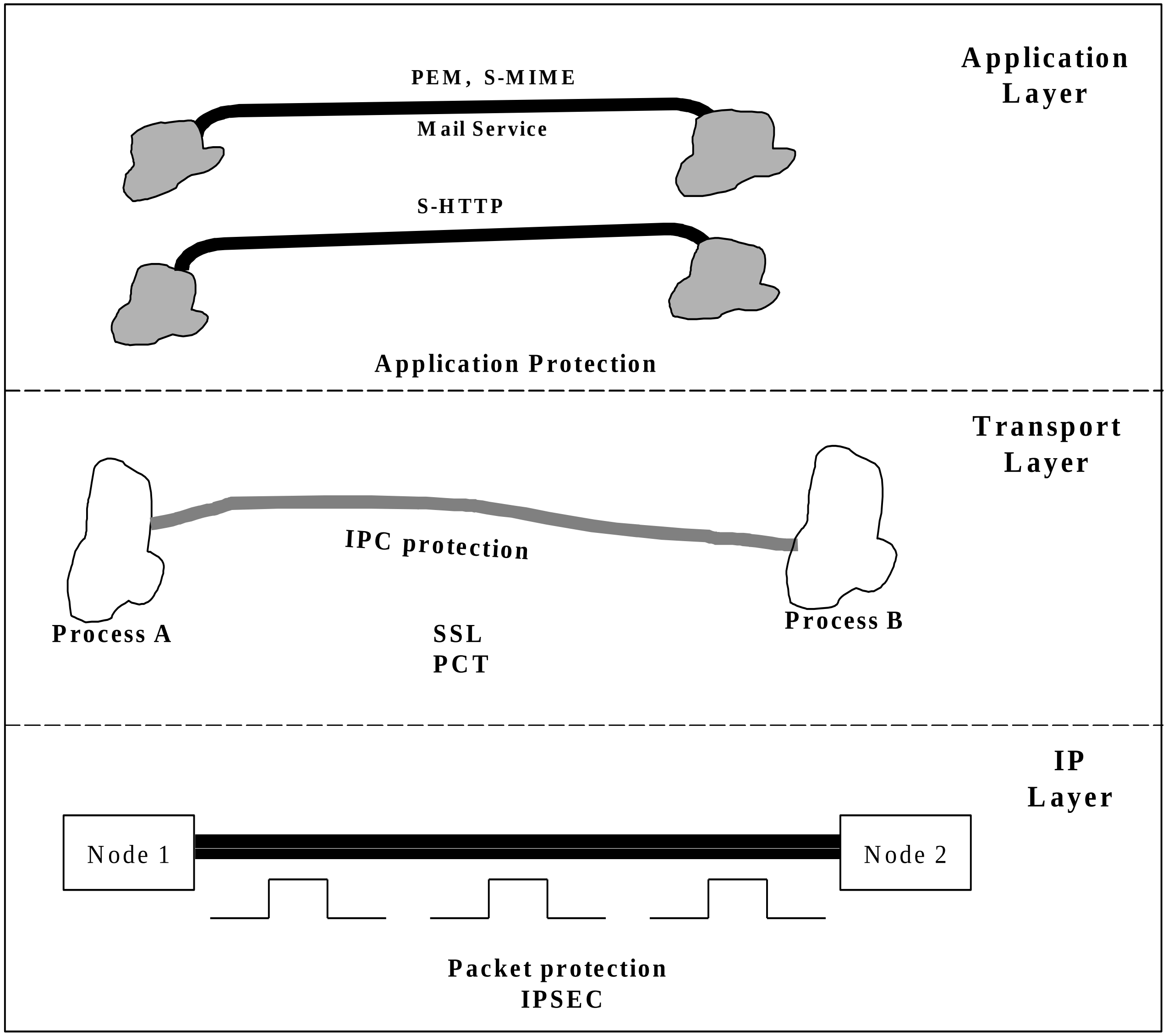
Patterns for secure process management



The network layers

- ***Firewalls***
- ***Intrusion Detection Systems***
- ***Network layer defenses***

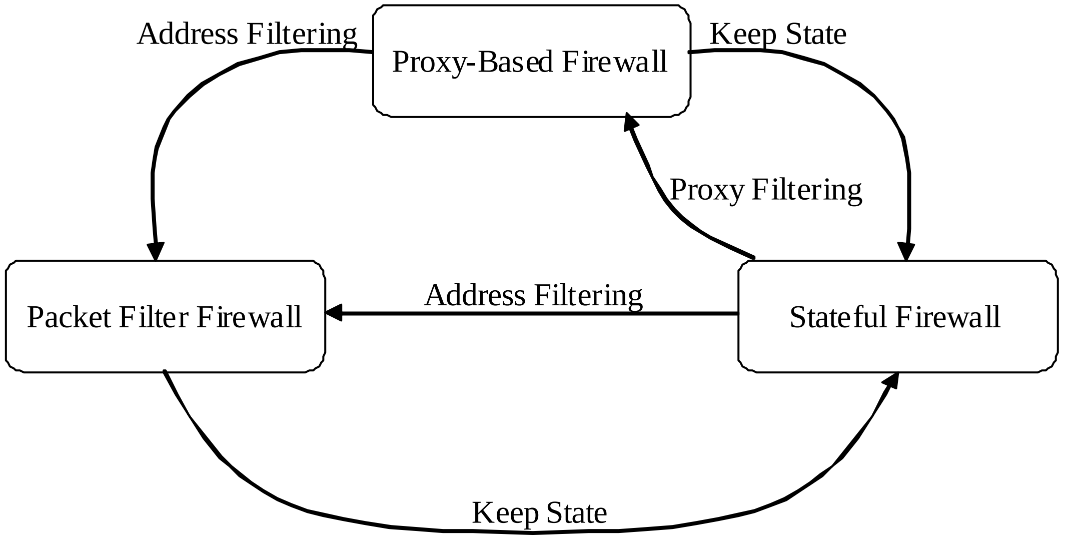
Secure channels



Patterns for firewalls

- ***Packet Filter Firewall*****. *Filter incoming and outgoing network traffic in a computer system based on network addresses.***
- ***Application/Proxy Firewall* *. Inspect (and filter) incoming and outgoing network traffic based on the type of application they are accessing.***
- ***Stateful firewall* *Filter incoming and outgoing network traffic in a computer system based on network addresses and the state information derived from past communications.***

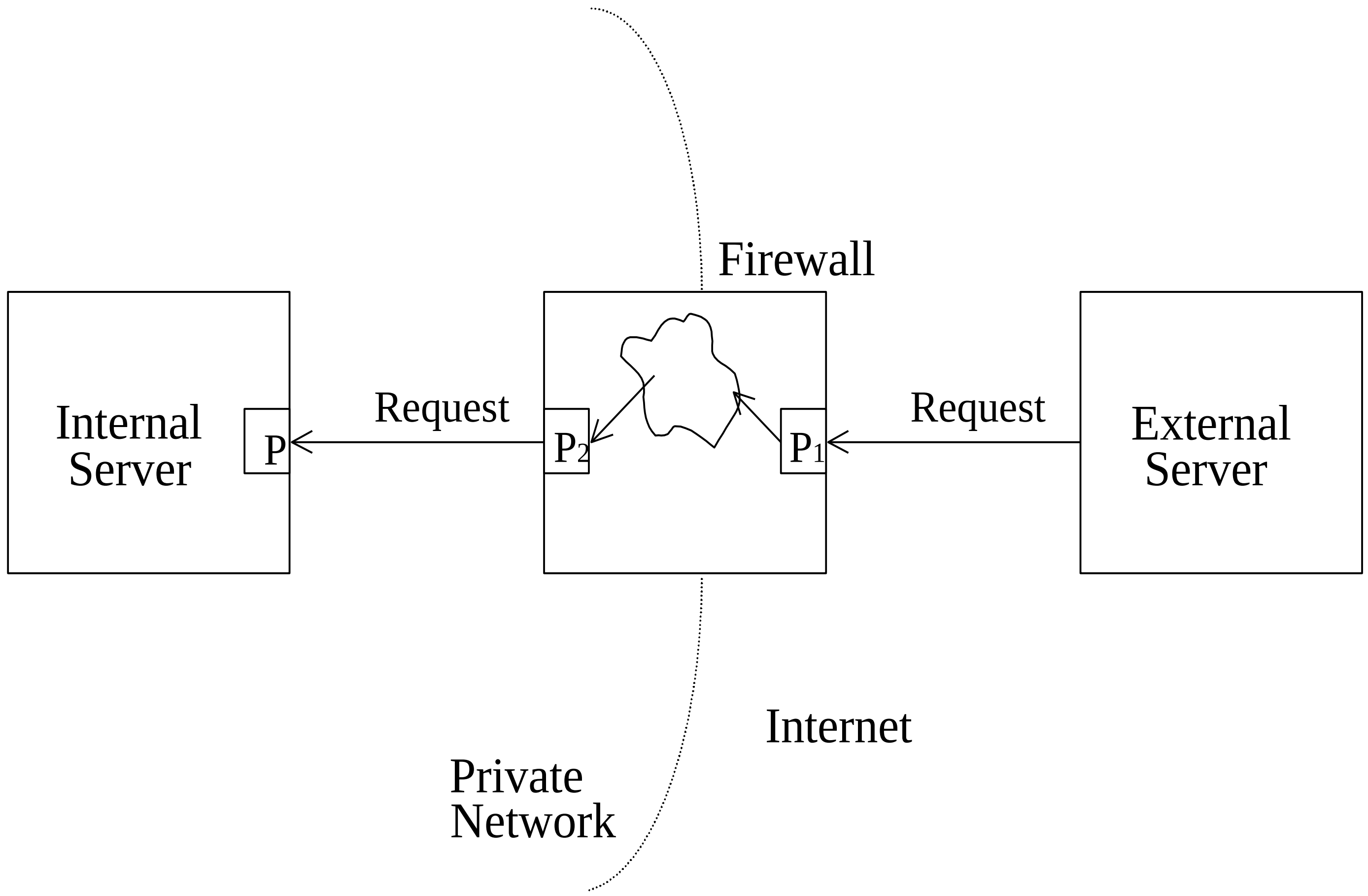
Firewall patterns



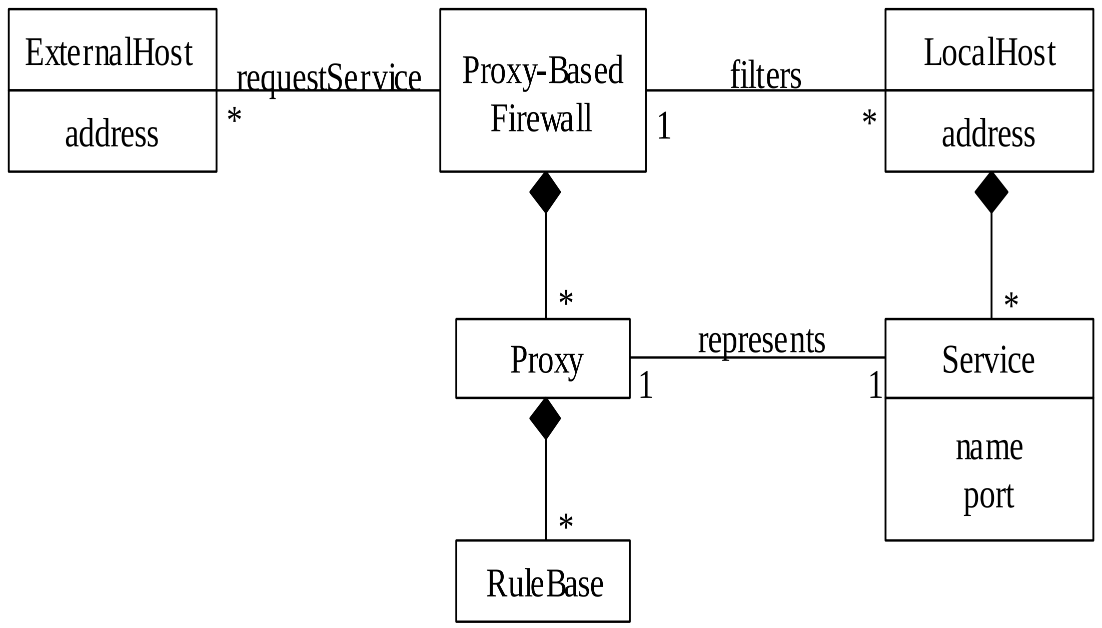
Application layer (proxy) firewall

- ***Uses security proxies to represent services***
- ***A variety of the Proxy pattern***
- ***Prevents direct access***
- ***Analyzes application commands***
- ***Keeps logs for later auditing***

Application layer firewall



Proxy-based firewall

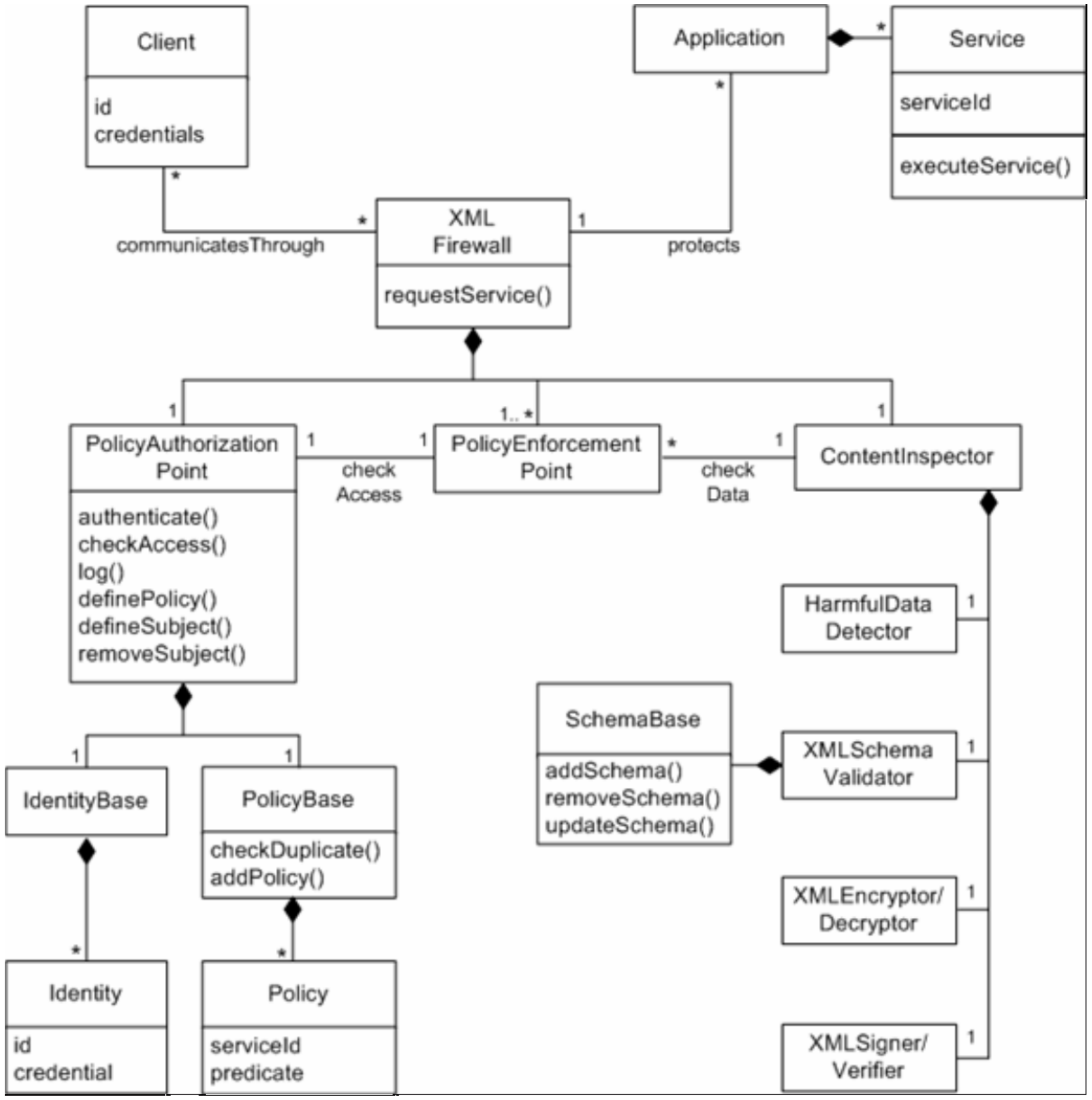


In summary

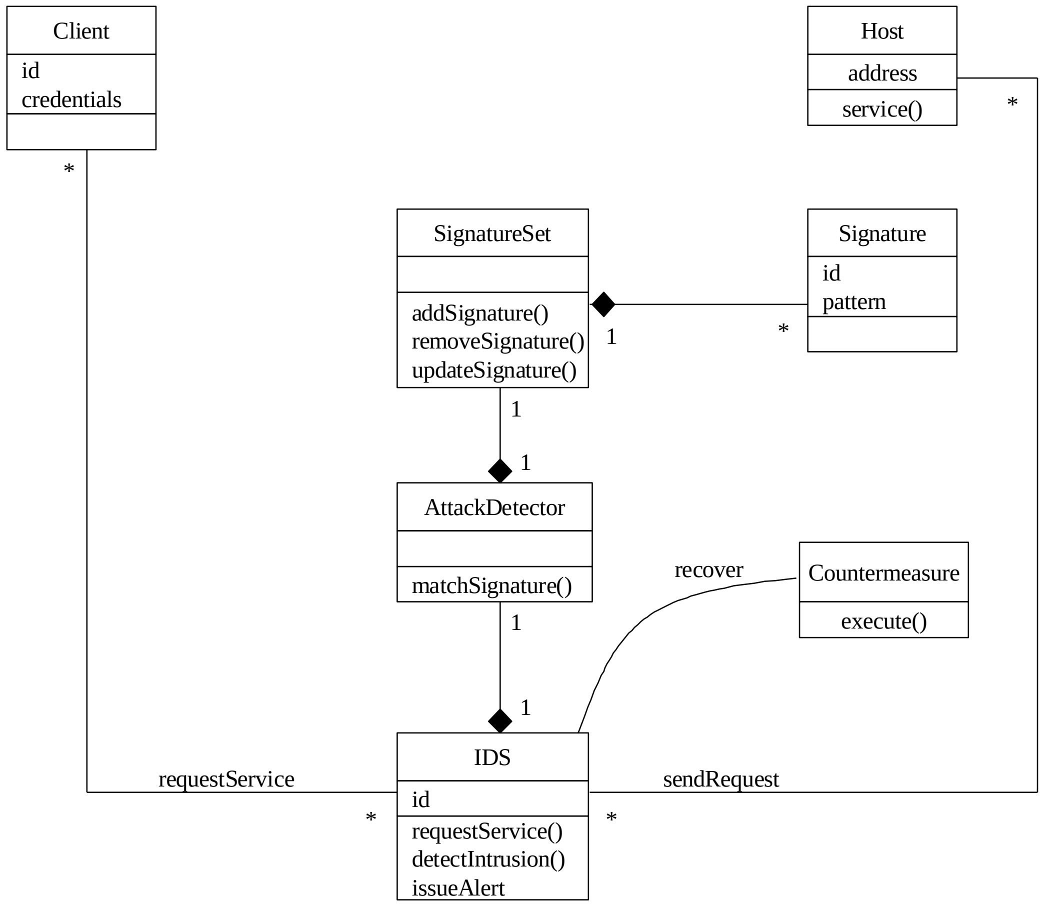
- ***Firewalls are examples of the Reference Monitor pattern applying a simple Access Matrix model***
- ***Packet filter and proxy firewalls complement each other***
- ***Can be further complemented with intrusion detection***

XML firewall

- ***Controls input/output of XML applications***
- ***Well-formed documents (schema as reference)***
- ***Harmful data (wrong type or length)***
- ***Encryption/decryption***
- ***Signed documents***


Client
id
credentials
Application
Service
serviceId
executeService()
XML Firewall
requestService()
communicatesThrough
protects
PolicyAuthorization Point
authenticate()
checkAccess()
log()
definePolicy()
defineSubject()
removeSubject()
check Access
PolicyEnforcement Point
check Data
ContentInspector
HarmfulData Detector
SchemaBase
addSchema()
removeSchema()
updateSchema()
XMLSchema Validator
XMLEncryptor/ Decryptor
XMLSigner/ Verifier
IdentityBase
PolicyBase
checkDuplicate()
addPolicy()
Identity
id
credential
Policy
serviceId
predicate

Knowledge-based IDS pattern



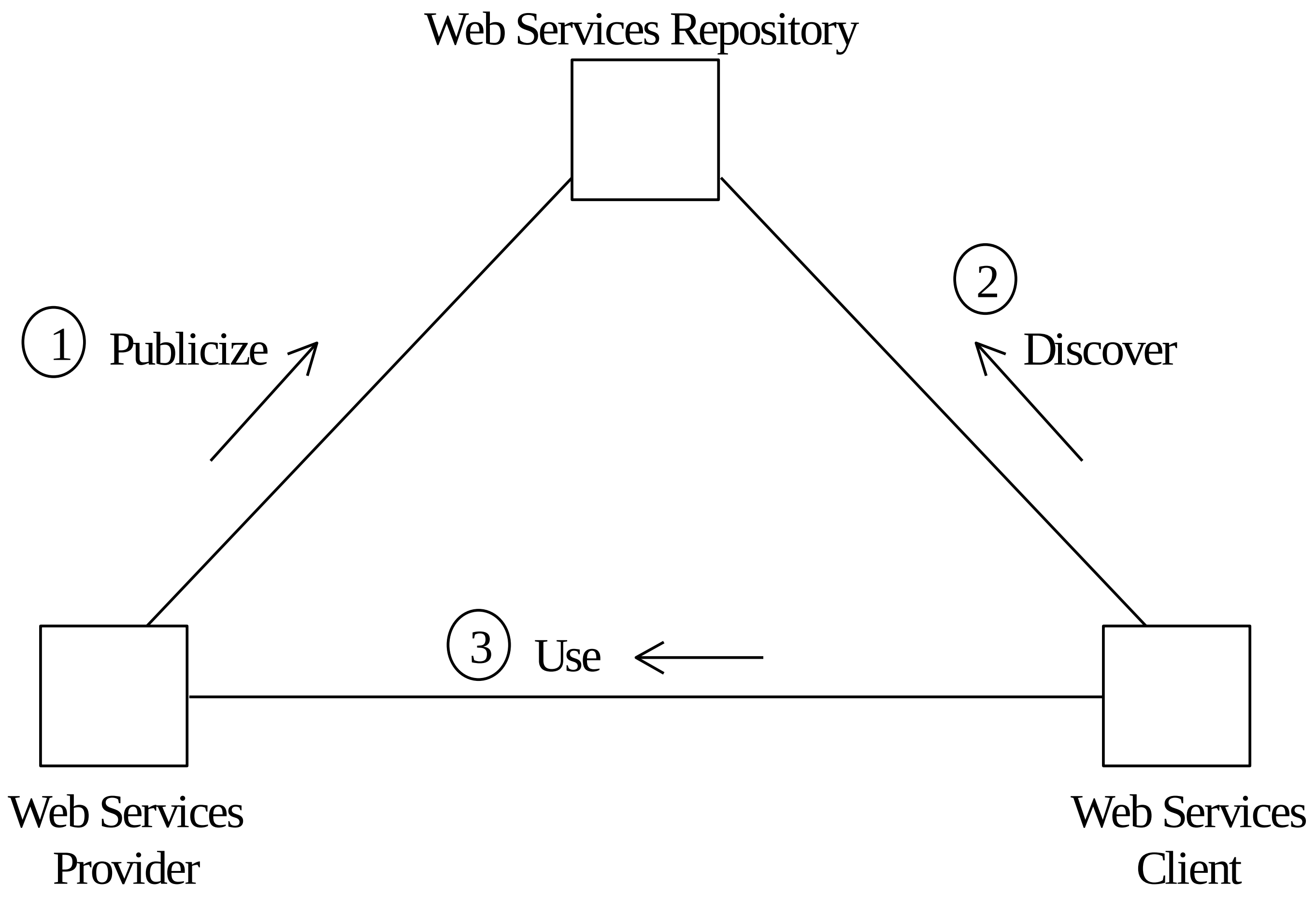
Distribution and web services

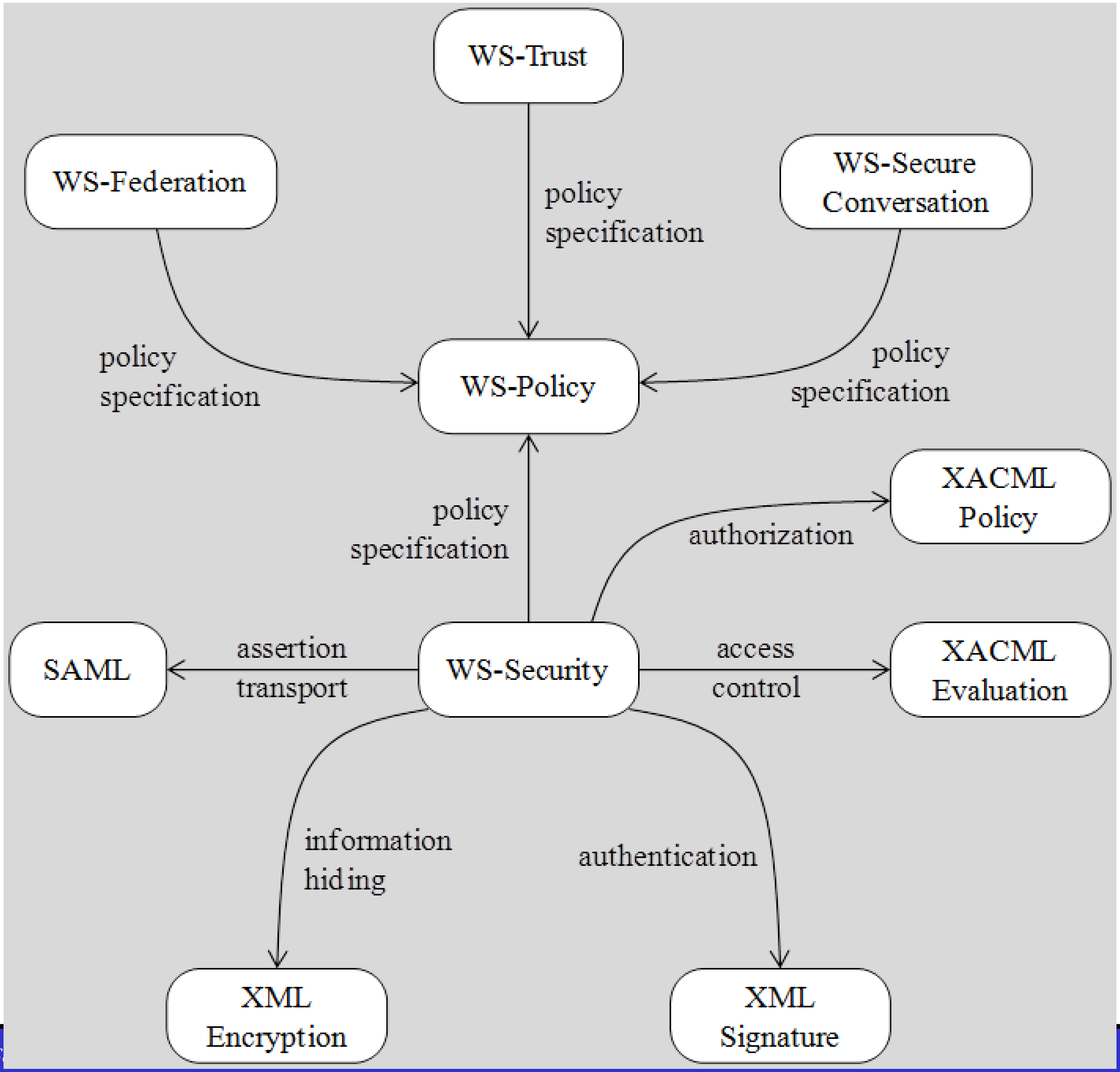
- ***Approaches***
- ***Authentication***
- ***Distribution***
- ***Identity***
- ***Web services***
- ***Cloud Computing***

Web services security

- *Application Firewall* ***[Del04]. The application firewall filters calls and responses to/from enterprise applications, based on an institution access control policies.***
- *XML Firewall* ***[Del04]. Filter XML messages to/from enterprise applications, based on business access control policies and the content of the message.***
- *XACML Authorization* ***[Del05]. Enable an organization to represent authorization rules in a standard manner.***
- *XACML Access Control Evaluation* ***[Del05]. This pattern decides if a request is authorized to access a resource according to policies defined by the XACML Authorization pattern. .***
- *WSPL* ***[Del05]. Enable an organization to represent access control policies for its web services in a standard manner. It also enables a web services consumer to express its requirements in a standard manner.***

Use of web services

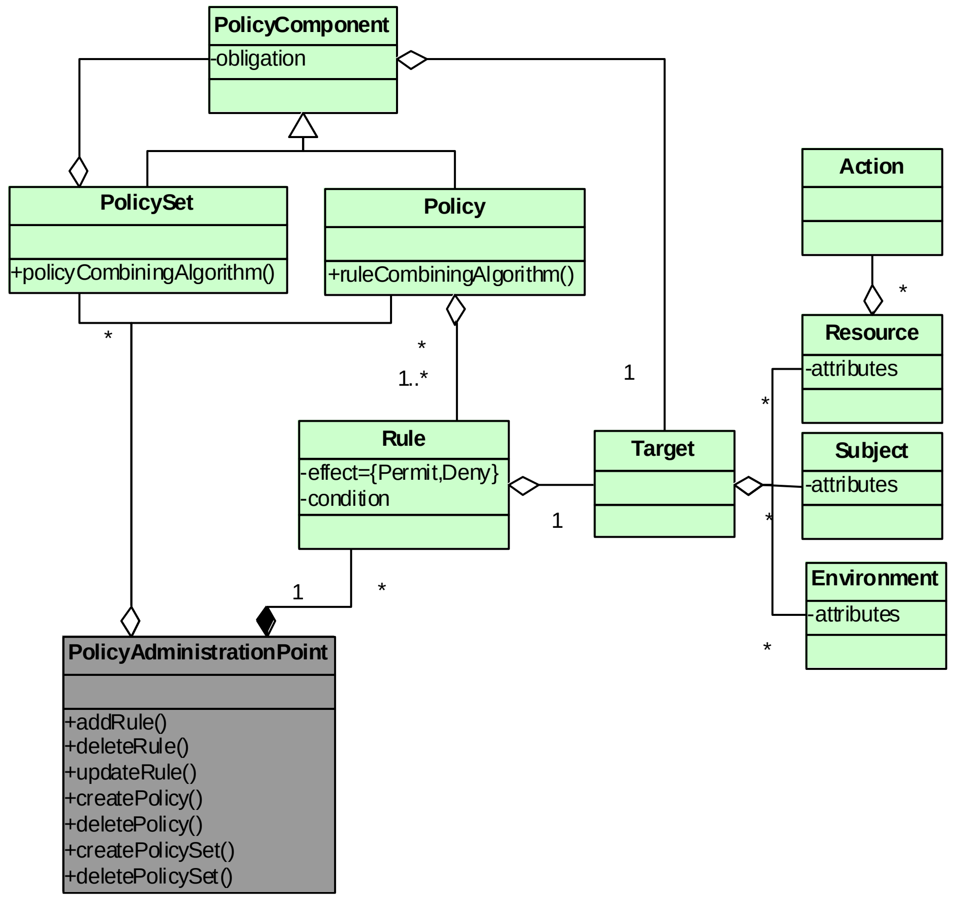



WS-Trust
WS-Federation
WS-Secure Conversation
policy specification
policy specification
policy specification
WS-Policy
policy specification
XACML Policy
authorization
SAML
assertion transport
WS-Security
access control
XACML Evaluation
information hiding
authentication
XML Encryption
XML Signature

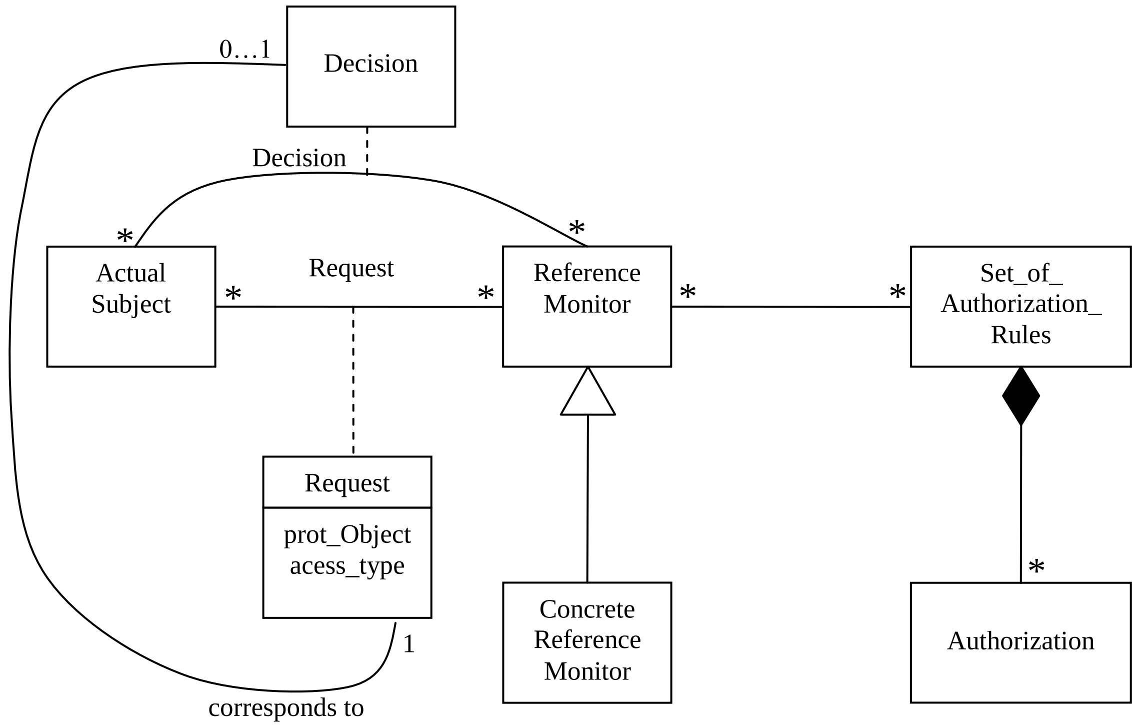
XACML

- ***Special technical committee of OASIS***
- ***Specification of policies for information access over the Internet and their enforcement***
- ***Combines work of IBM Tokyo and University of Milano, Italy.***
- ***Implemented by Sun in early 2003***

XACML Authorization



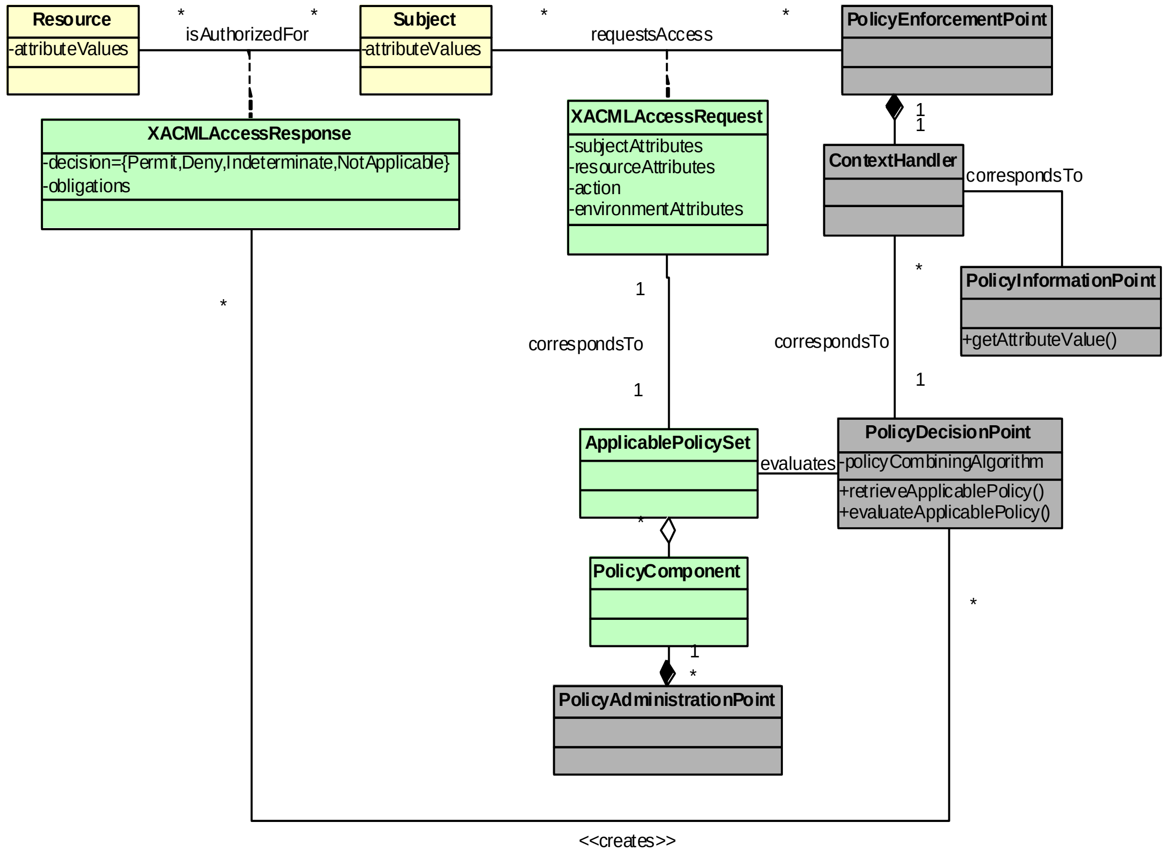
Reified Reference Monitor



RRM

- ***In the RRM the decision, instead of being 'yes' or 'no', is a class to which we can apply operations (reification)***
- ***There is zero or one decision per request***
- ***Information about the context or any other information can be used to produce the decision***

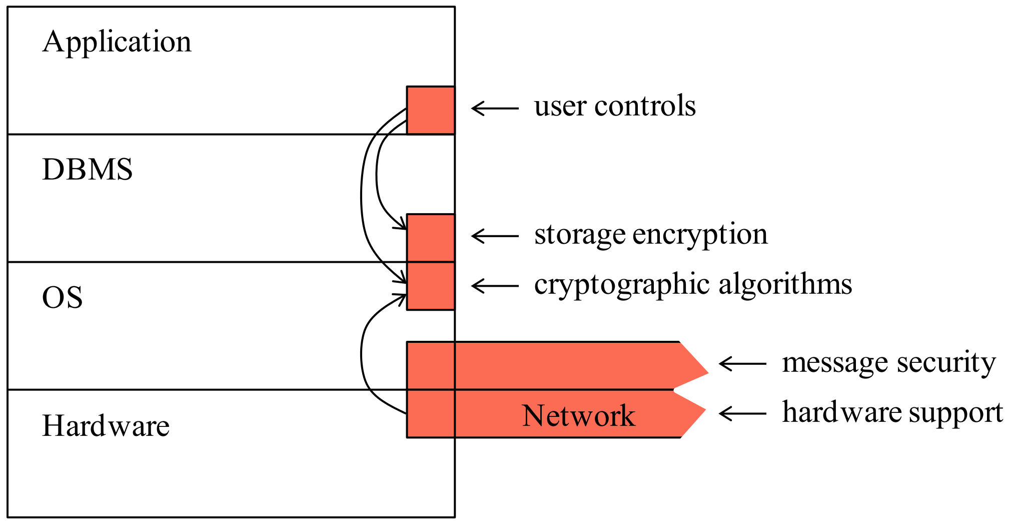
Access control evaluation



Abstraction in the use of patterns

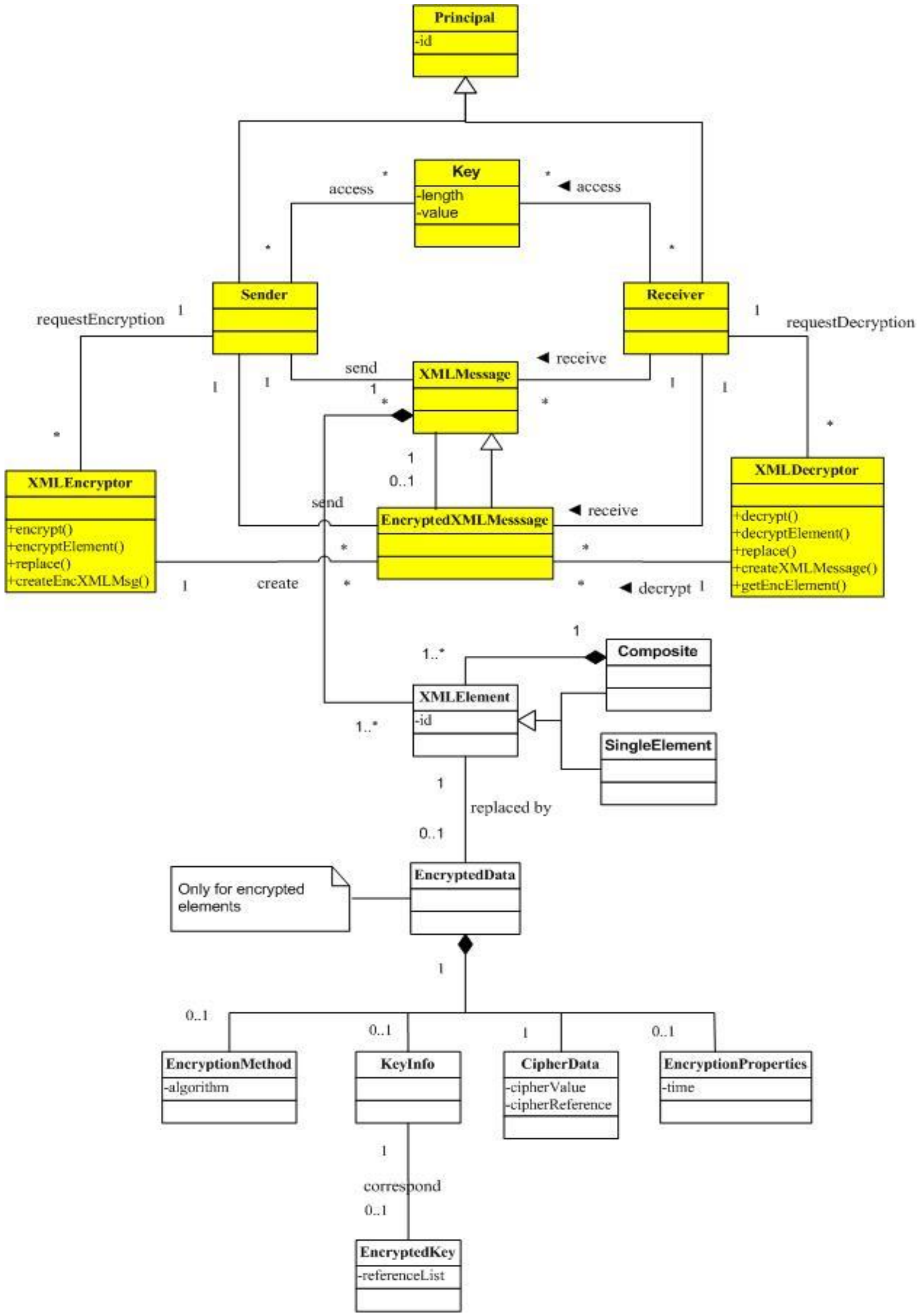
- ***We can see the Composite and Authorization patterns in XACML Authorization***
- ***We can see the Reference Monitor in XACML Evaluation***
- ***Abstraction helps model understanding***

Use of encryption in the architecture

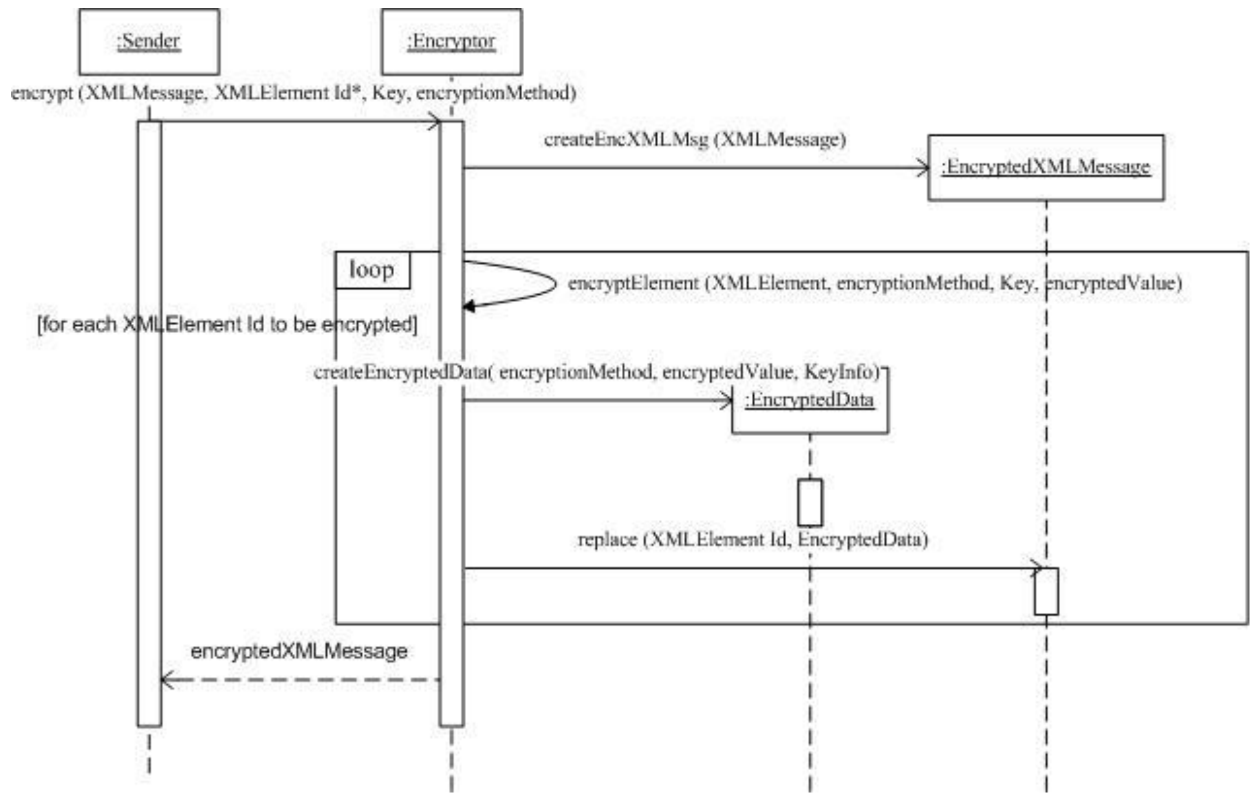


XML Encryption

- ***The XML Encryption standard describes the syntax to represent XML encrypted data and the process of encryption and decryption. XML Encryption provides confidentiality by hiding selected sensitive information in a message using cryptography.***


Principal
-id
Key
-length
-value
access
access
Sender
Receiver
requestEncryption
requestDecryption
send
receive
XMLMessage
XMLEncryptor
+encrypt()
+encryptElement()
+replace()
+createEncXMLMsg()
XMLDecryptor
+decrypt()
+decryptElement()
+replace()
+createXMLMessage()
+getEncElement()
EncryptedXMLMesssage
send
receive
create
decrypt
Composite
XMLElement
-id
SingleElement
replaced by
EncryptedData
Only for encrypted elements
EncryptionMethod
-algorithm
KeyInfo
CipherData
-cipherValue
-cipherReference
EncryptionProperties
-time
correspond
EncryptedKey
-referenceList

Encrypting elements


:Sender
:Encryptor
encrypt (XMLMessage, XMLElement Id*, Key, encryptionMethod)
createEncXMLMsg (XMLMessage)
:EncryptedXMLMessage
loop
encryptElement (XMLElement, encryptionMethod, Key, encryptedValue)
[for each XMLElement Id to be encrypted]
createEncryptedData(encryptionMethod, encryptedValue, KeyInfo)
:EncryptedData
replace (XMLElement Id, EncryptedData)
encryptedXMLMessage

The design of secure systems

- We not only have to satisfy functional specs but nonfunctional specs.
- Security is a nonfunctional aspect
- We cannot show absence of security flaws
- We must use good development methods and hope for the best
- Add-on security is not very secure

Use patterns at all levels

- ***Patterns for models define the highest level***
- ***At each lower level we refine the model patterns to consider the specific aspects of each level***
- ***Patterns for file systems, web documents, cryptography, distributed objects, J2EE components***

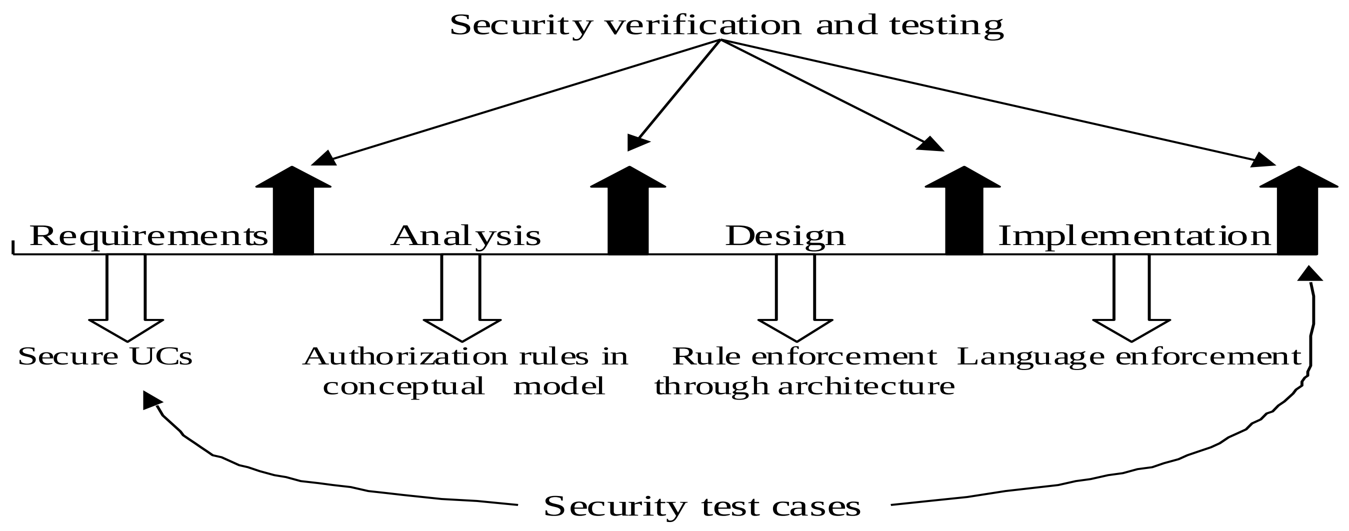
How to apply the patterns?

- ***A good catalog and classifications of patterns help a designer select among alternatives.***
- ***However, there is still the problem of when to apply a pattern during system development***
- ***We need some systematic approach to decide when we need to use a pattern, a secure systems methodology***

Security principles for application design

- *Security constraints must be defined at the highest layer, where their semantics are clear, and propagated to the lower levels, which enforce them.*
- *All the layers of the architecture must be secure.*
- *We can define patterns at all levels. This allows a designer to make sure that all levels are secured, and also makes easier propagating down the high-level constraints.*
- *We must apply security at all development stages (tactic-based approaches start from design)*

Security along the life cycle


Security verification and testing
Requirements
Analysis
Design
Implementation
Secure UCs
Authorization rules in conceptual model
Rule enforcement through architecture
Language enforcement
Security test cases

A methodology for secure systems design I

- ***Domain analysis stage*: A business model is defined. Legacy systems are identified and their security implications analyzed. Domain and regulatory constraints are identified. Policies must be defined up front, in this phase.**
- ***Requirements stage*: Use cases define the required interactions with the system. Applying the principle that security must start from the highest levels, it makes sense to relate attacks to use cases. We study each action within a use case and see which threats are possible. We then determine which policies would stop these attacks. From the use cases we can also determine the needed rights for each actor and thus apply a need-to-know policy.**

Requirements stage

- *Use cases are determined*
- *Activity diagrams for use cases or sequences of use cases*
- *Define level of security needed*
- *Identify attacks*
- *Select attacks based on risk analysis*

Identifying attacks

- ***We need to know what kind of attacks to expect.***
- ***We relate attacks to attacker goals***
- ***We study systematically all the possible attacks to each activity in a use case***
- ***Use cases define all functional interactions***

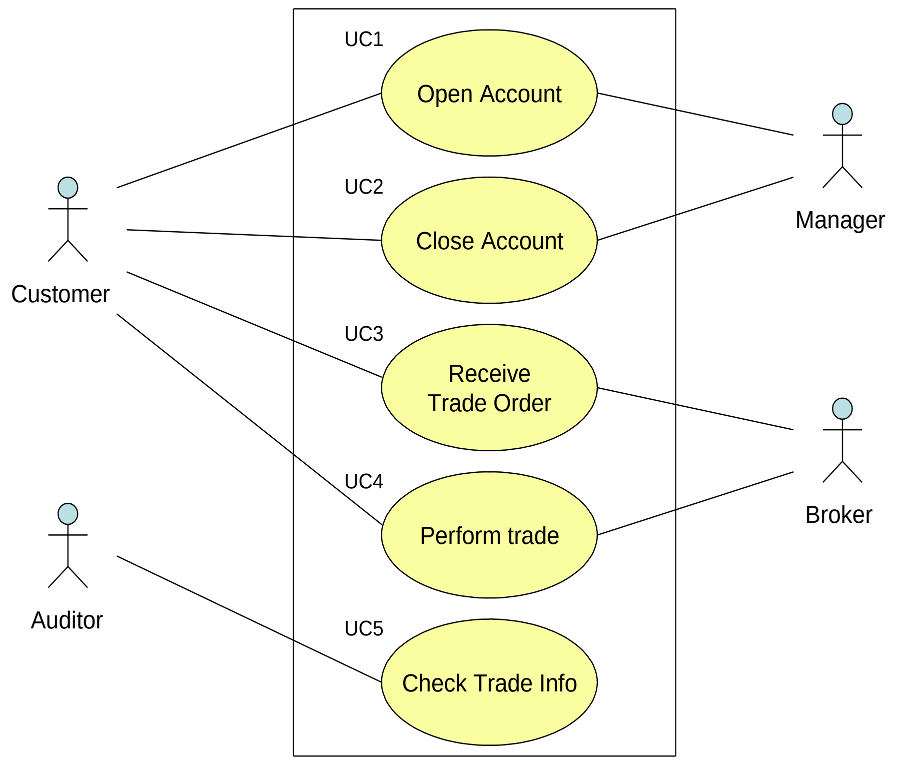
Attacker goals

- *Attacker is not interested in changing a few bits or destroying a message*
- *Attacker wants to accomplish some objective, e.g., steal money, steal identity*
- *This is applying the principle of defining security at the semantic levels*
- *We also need to comply with standards*

Example: Applying security policies to stop threats

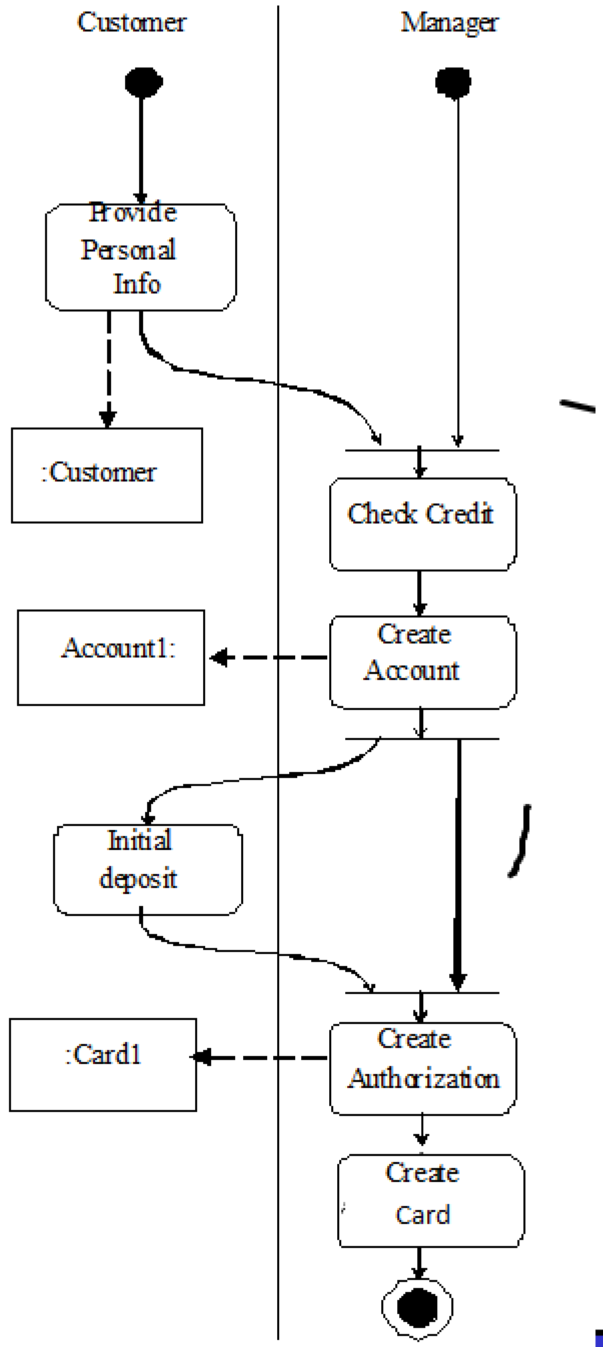
- ***Consider a financial company that provides investment services to its customers. Customers can open and close accounts in person or through the Internet. Customers who hold accounts can send orders to the company for buying or selling commodities (stocks, bonds, real estate, art, etc.). Each customer account is in the charge of a custodian (a broker), who carries out the orders of the customers. Customers send orders to their brokers by email or by phone. A government auditor visits periodically to check for application of laws and regulations.***

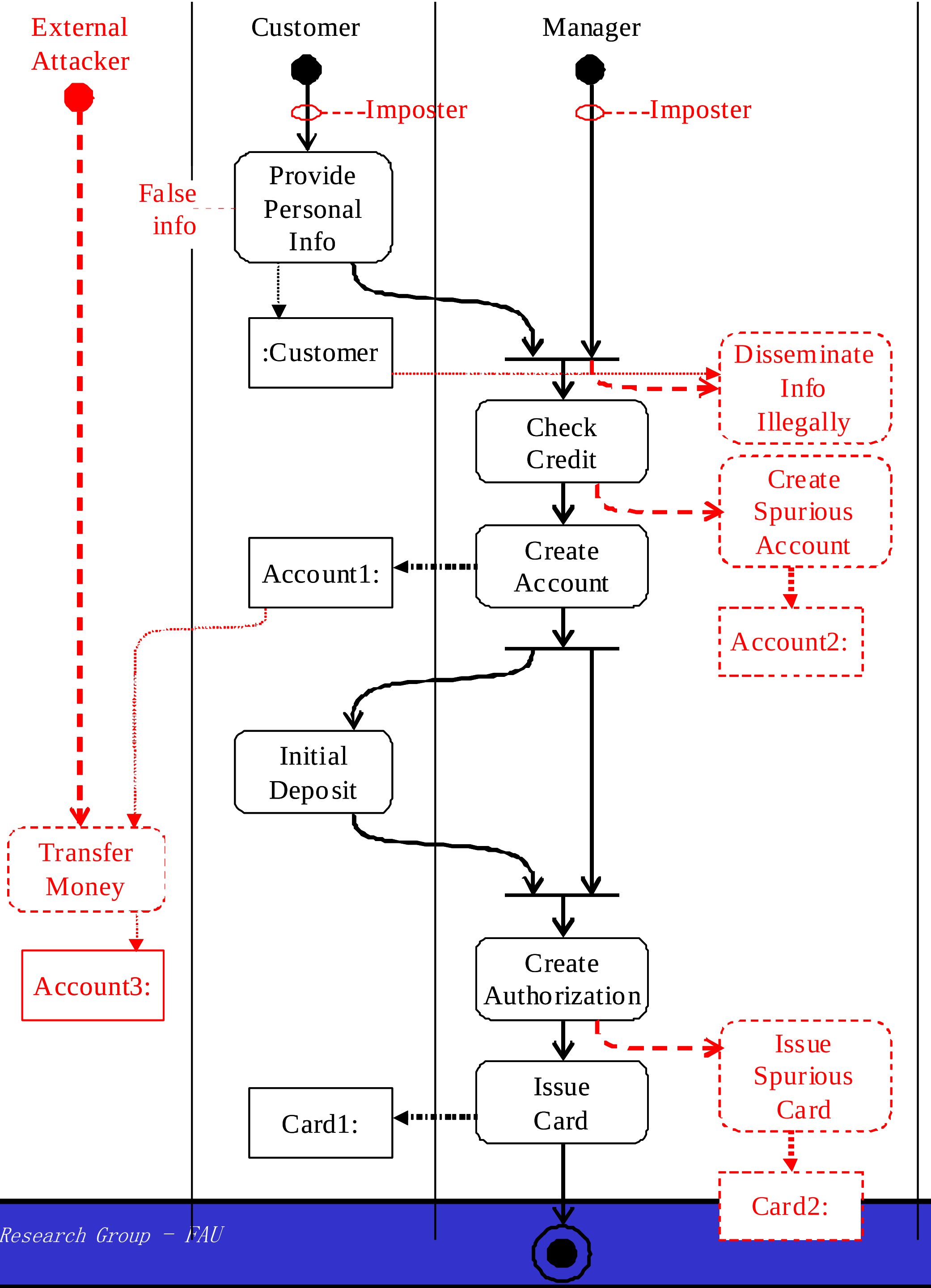
A financial institution



From threats to policies

- ***A systematic way to identify system threats, and determining policies to stop and/or mitigate their effects***
- ***Analysis of the flow of events in a use case or a group of use cases, in which each activity is analyzed to uncover related threats. This analysis should be performed for all the system uses cases***
- ***Then we select appropriate security policies which can stop and/or mitigate the identified threats.***


Customer
Manager
Provide Personal Info
:Customer
Check Credit
Create Account
Account1:
Initial deposit
Create Authorization
:Card1
Create Card


External Attacker
Customer
Manager
Imposter
Imposter
False info
Provide Personal Info
:Customer
Disseminate Info Illegally
Check Credit
Create Spurious Account
Create Account
Account1:
Account2:
Initial Deposit
Transfer Money
Account3:
Create Authorization
Issue Spurious Card
Issue Card
Card1:
Card2:

Threats

- ***T1.The customer is an impostor and opens an account in the name of another person***
- ***T2.The customer provides false information and opens an spurious account***
- ***T3.The manager is an impostor and collects data illegally***
- ***T4.The manager collects customer information to use illegally***
- ***T5.The manager creates a spurious account with the customer's information***
- ***T6.The manager creates a spurious authorization card to access the account***
- ***T7.An attacker tries to prevent the customers to access their accounts***
- ***T8.An attacker tries to move money from an account to her own account***

Misuses for this use case

- ***Use Cases 1 and 2: Customer is an impostor. Possible confidentiality and integrity violations.***

  Manager impersonation. Can capture customer information. Confidentiality attack.
- ***Use Cases 3 and 4. Broker impersonation. Possible confidentiality violation.***

  Customer can deny giving a trade order. Repudiation. Customer impersonation. Confidentiality or integrity attacks.

  Stockbroker embezzling customer's money.
- ***Use Case 5. Impersonation of auditor. Confidentiality violation.***

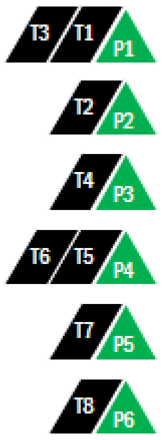
Use case analysis leads to policies

- ***T1. T3. Mutual authentication. Every interaction across system nodes is authenticated.***
- ***T2. Verify source of information.***
- ***T4. Logging. Since the manager is using his legitimate rights we can only log his actions for auditing at a later time.***
- ***T5. T6. Separation of administration from use of data. For example, a manager can create accounts but should have no rights to withdraw or deposit in the account.***
- ***T7. Protection against denial of service. We need some redundancy in the system to increase its availability.***
- ***T8. Authorization. If the user is not explicitly authorized he should not be able to move money from any account.***

Policies can be realized with patterns

Systematic mapping of threats to policies

Threats	
T1	The customer is an impostor and opens and account in the name of another person
T2	The customer provides false info (financial, address) to an account
T3	The manager is an impostor and collects data illegally
T4	The manager collects customer info to use illegally
T5	The manager creates a spurious account with the customer's info
T6	The manager creates a spurious authorization card to access the account
T7	An attacker tries to prevent the customers access to their accounts (denial of service)
T8	An attacker tries to move money from an account to her own account



Policies	
P1	Mutual authentication
P2	Verify source of information
P3	Logging
P4	Separation of duties
P5	Protection against denial of service (redundancy, intrusion detection system, etc.)
P6	Authorization

a) Threat list

b) Avoidance and/or Mitigation

c) Policy list

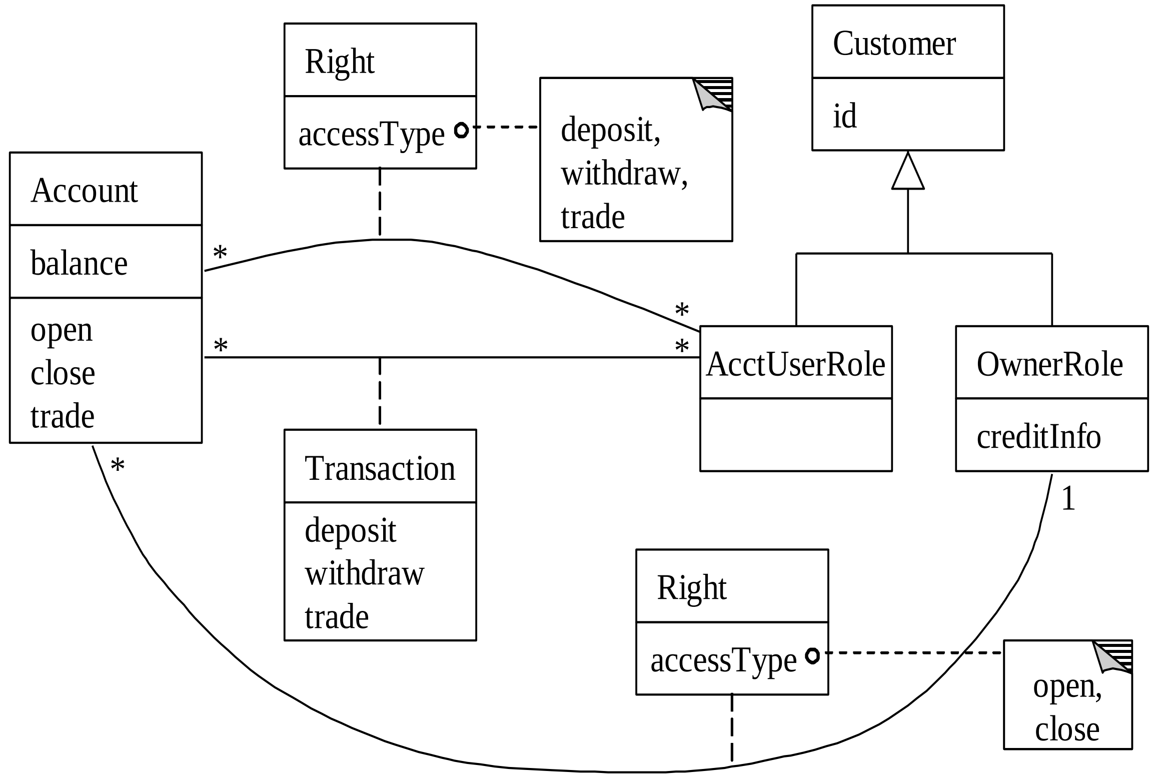
Analysis stage

- ***Analysis patterns can be used to build the conceptual model. Security patterns describe security models or mechanisms. We can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases.***

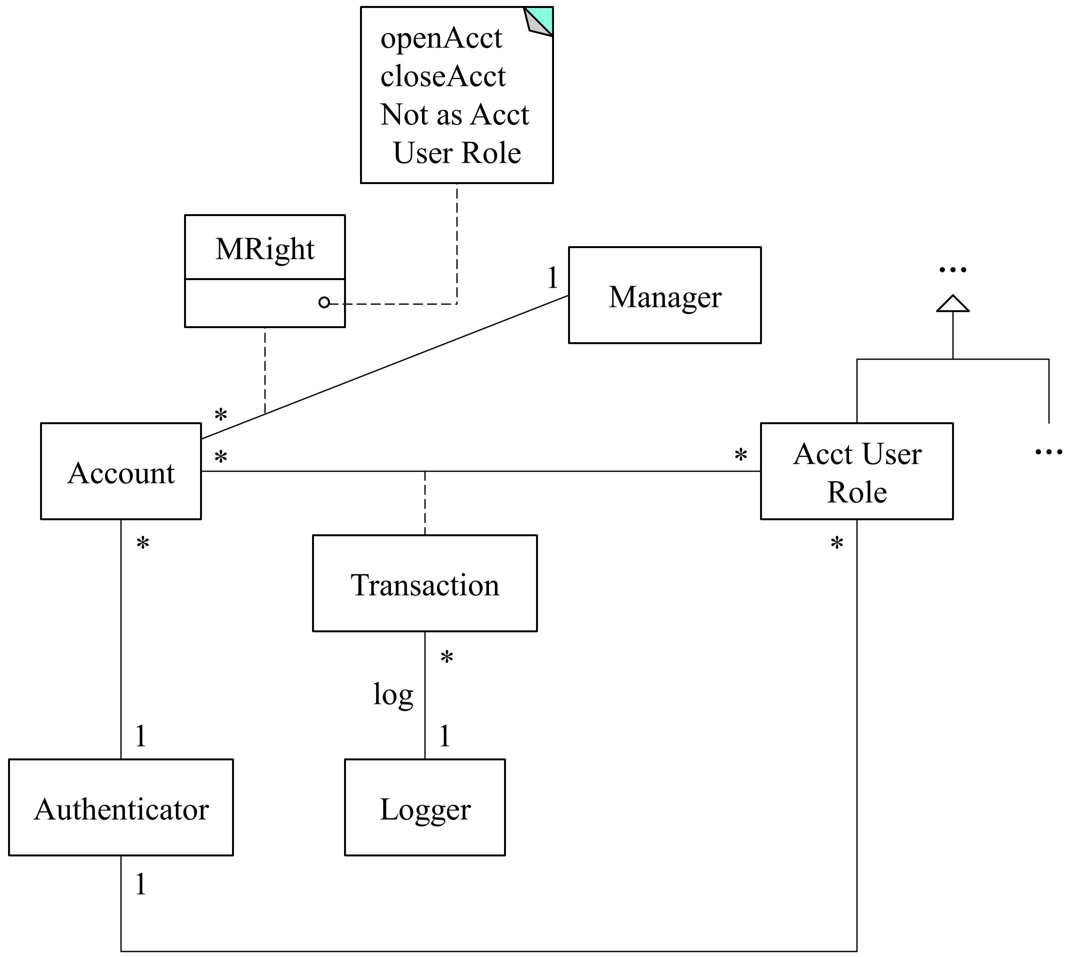
Role rights for financial institution

- ***Customers can open/close accounts***
- ***Customers can initiate trade***
- ***Broker can perform trade***
- ***Auditor can inspect (read) trade transactions***

Rights for financial application



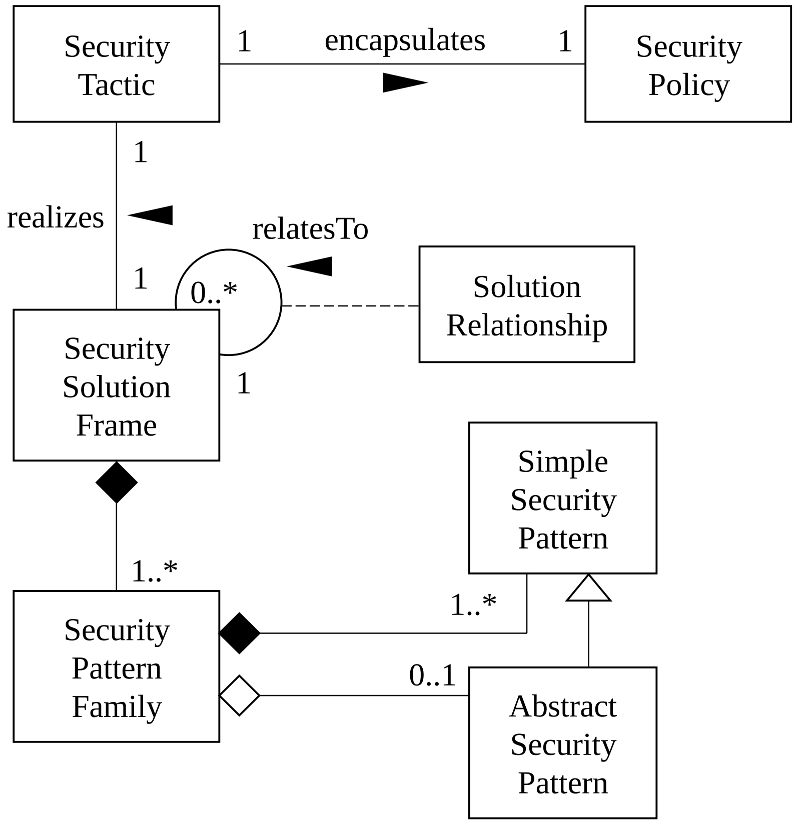
Adding more security: authentication, logging, separation of duty



Security Solution Frames

- ***Sets of related patterns that partition the solution space horizontally into separated but related concerns: Pattern Families, and vertically according to levels of abstraction.***
- ***Families treat one part needed to fully realize a tactic.***
- ***A pattern may belong to more than one family.***

SSF metamodel



Canonical abstraction levels

- ***Conceptual analysis*—contains one or more ASPs to describe the conceptual security solution**
- ***Abstract architecture*—contains patterns that determine a high-level, abstract architecture for the solution**
- ***Solution design*—contains patterns refining the abstract architecture using more concrete software components**
- ***Practical implementation details*—contains patterns about specialized aspects of some technology**

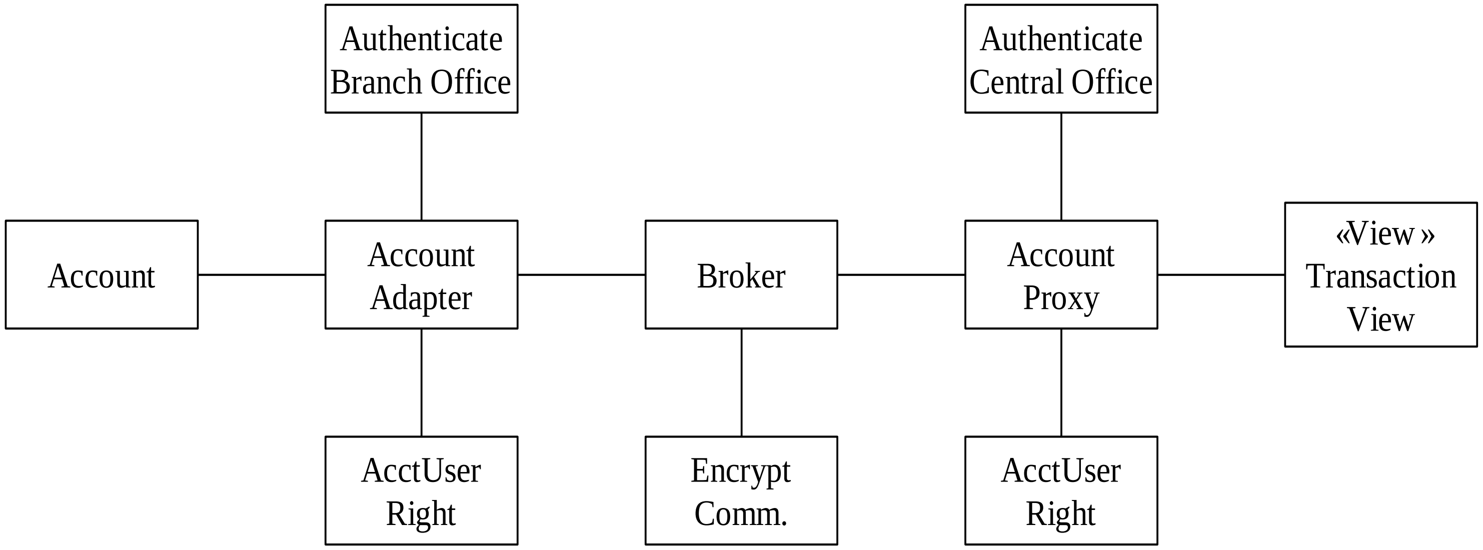
Design stage

- ***When we have the possible attacks to a system, design mechanisms are selected to stop these attacks. User interfaces should correspond to use cases and may be used to enforce the authorizations defined in the analysis stage. Secure interfaces enforce authorizations when users interact with the system. Components can be secured by using authorization rules for components. Distribution provides another dimension where security restrictions can be applied.***

Security is applied along the distribution path

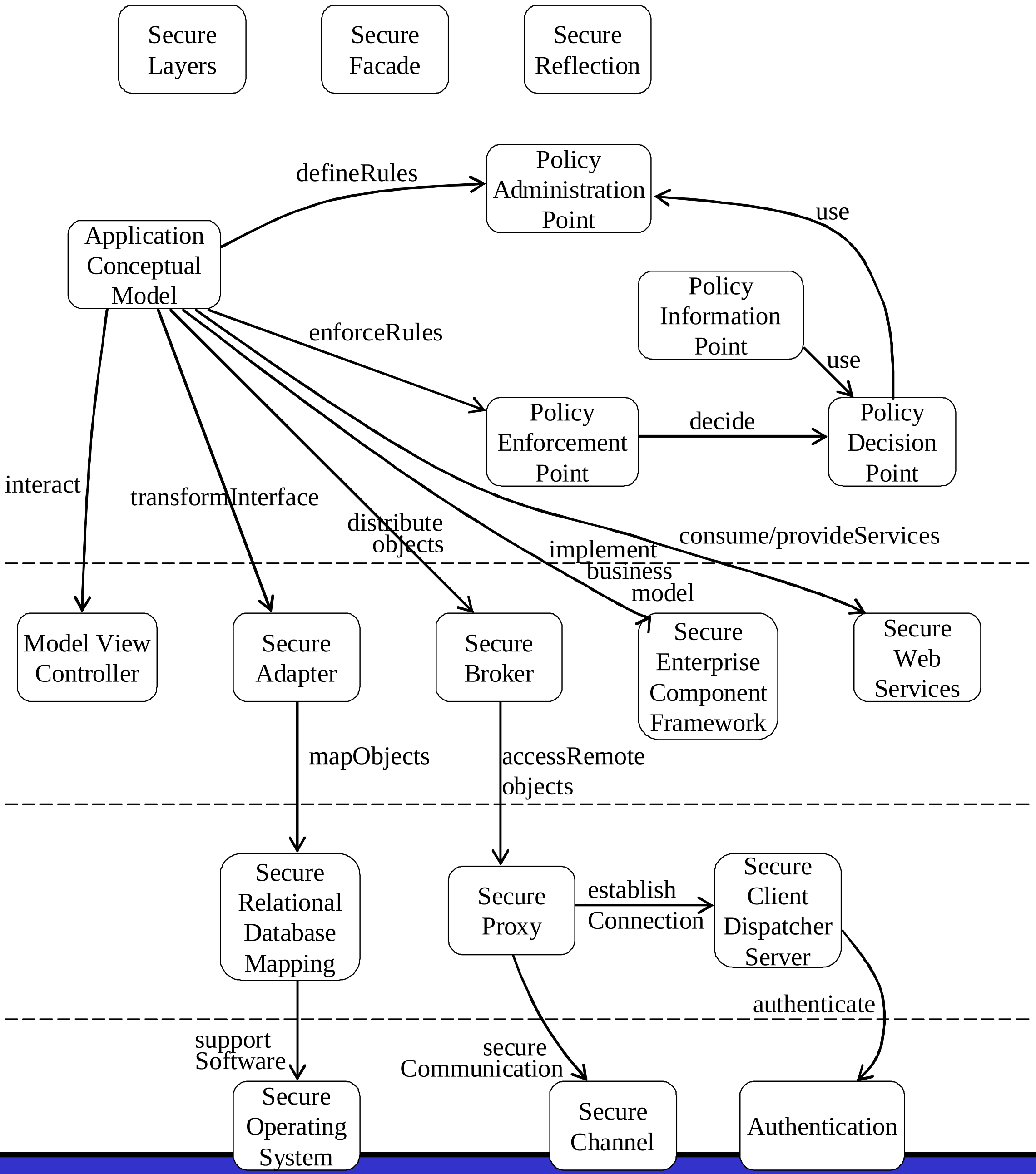
- ***Proxies and adapters apply authorization (Customer access to Accounts)***
- ***Proxies and adapters apply authentication (Branch office and central office mutual authentication)***
- ***Logging can be applied in the Transaction view (not shown)***
- ***Broker communications can be encrypted***
- ***Messages from Customers for trade can be signed***

Design model for financial application



Design stage patterns

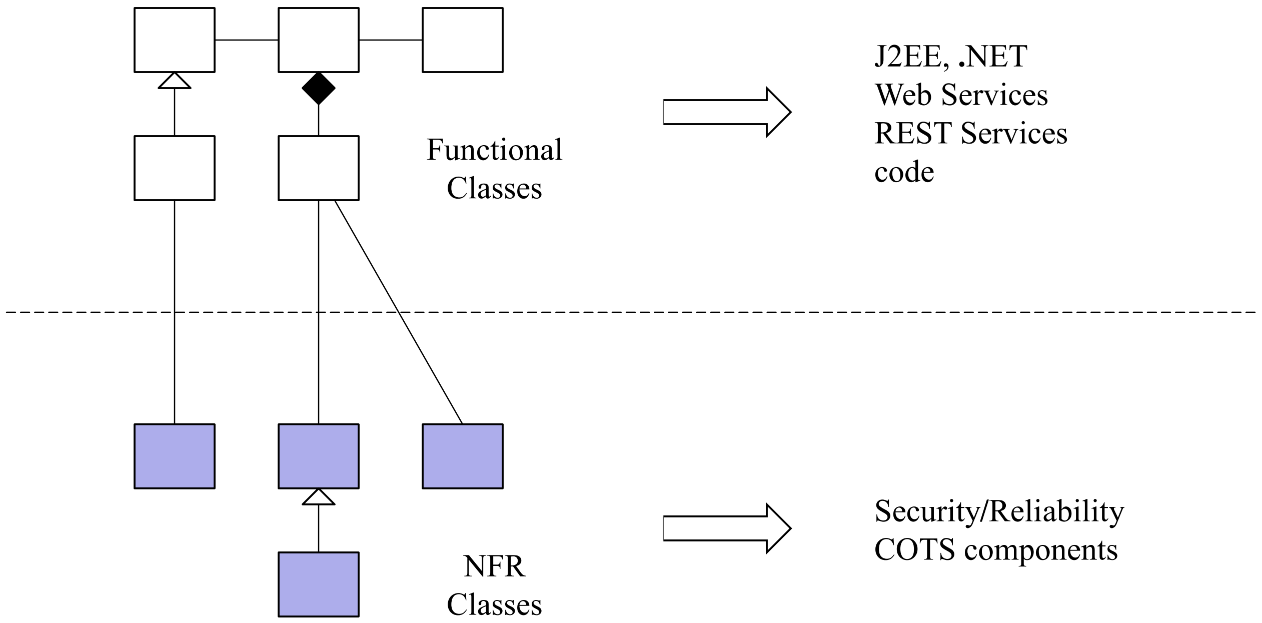
- ***We can map from the application level to the lower levels***
- ***Next diagram shows the application, distribution, database/communication, and operating system/communication levels***



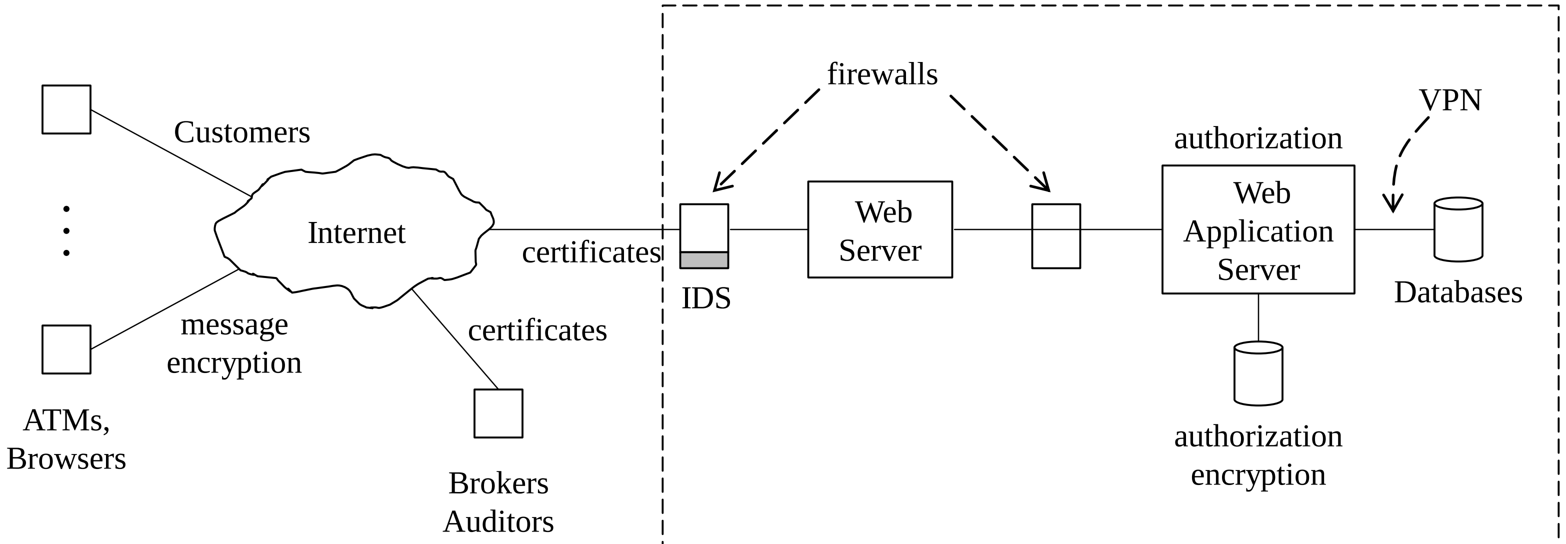
Secure Layers
Secure Facade
Secure Reflection
defineRules
Policy Administration Point
use
Application Conceptual Model
Policy Information Point
use
enforceRules
Policy Enforcement Point
decide
Policy Decision Point
interact
transformInterface
distribute objects
implement business model
consume/provideServices
Model View Controller
Secure Adapter
Secure Broker
Secure Enterprise Component Framework
Secure Web Services
mapObjects
accessRemote objects
Secure Relational Database Mapping
Secure Proxy
establish Connection
Secure Client Dispatcher Server
authenticate
support Software
secure Communication
Secure Operating System
Secure Channel
Authentication

Implementation stage

- ***Requires reflecting in the code the security rules defined in the design stage.***
- ***Because these rules are expressed as classes, associations, and constraints, they can be implemented as classes in object-oriented languages.***
- ***In this stage we can also select specific security packages or COTS components, e.g., a firewall product, a cryptographic package***
- ***Some of the patterns identified earlier in the cycle can be replaced by COTS components (these can be tested to see if they include a similar pattern).***


Functional
Classes
J2EE, .NET
Web Services
REST Services
code
NFR
Classes
Security/Reliability
COTS components

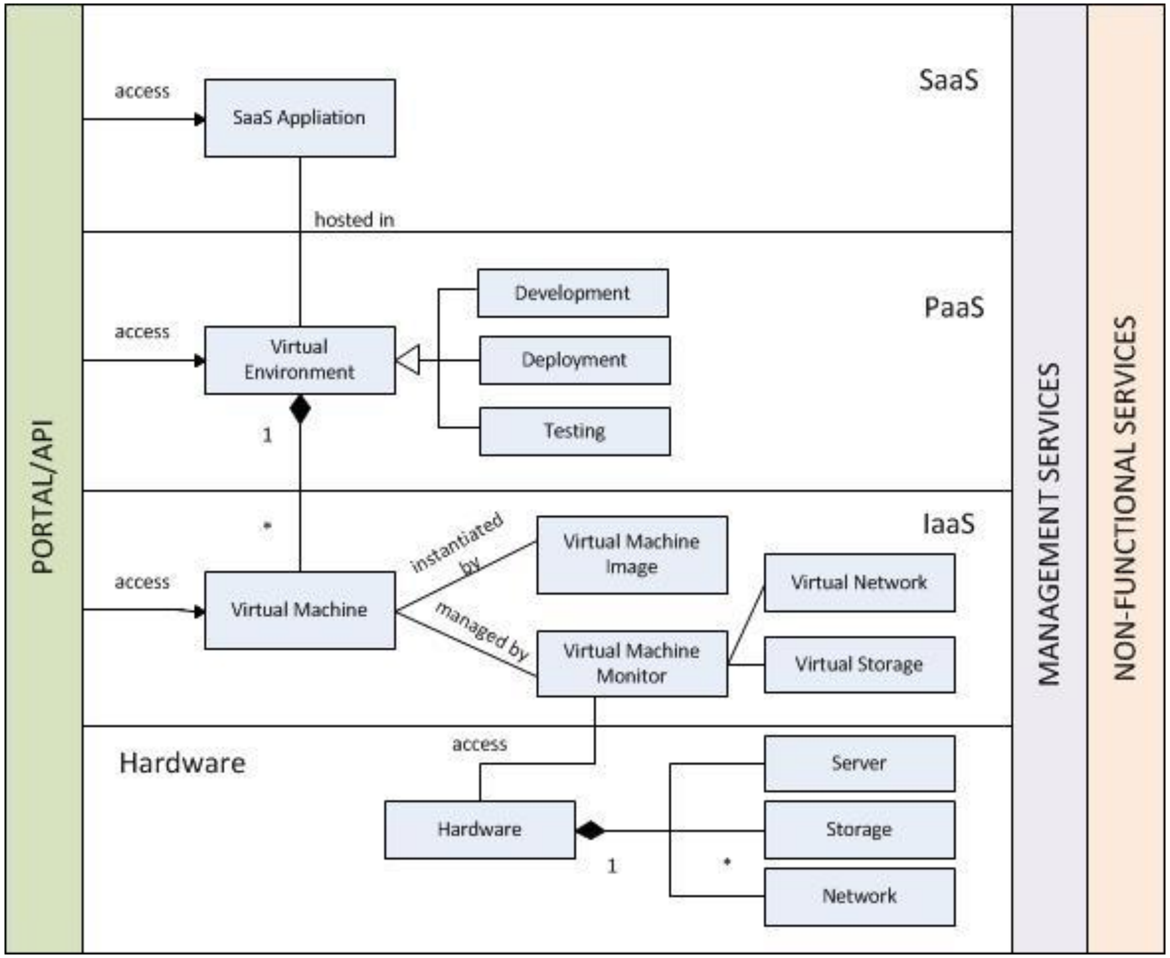
Deployment for financial institution



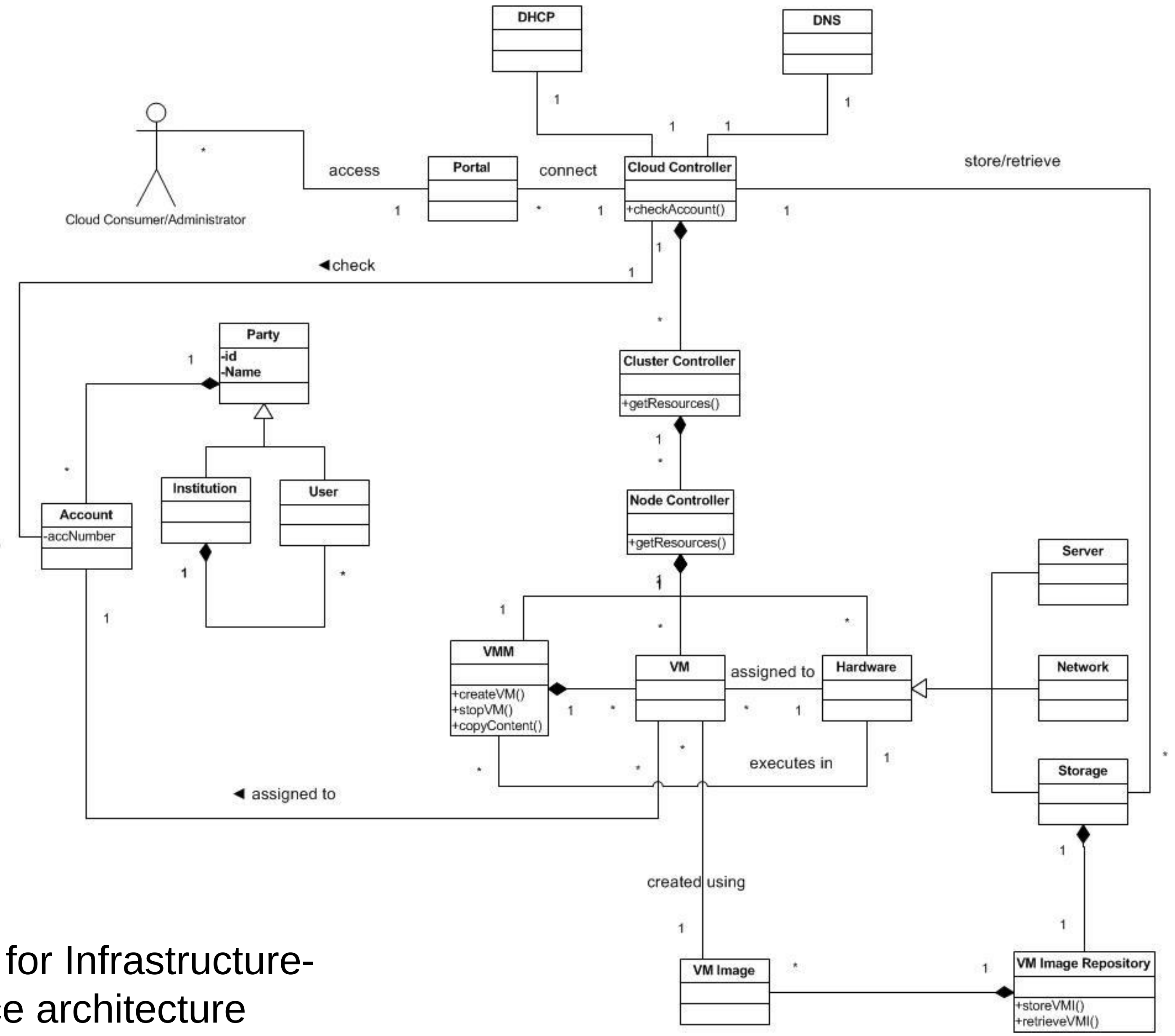
Reference architectures

- *A Domain Model (DM) is a model of an area of knowledge, e.g. financial systems, and has no software concepts*
- *A Reference Architecture (RA) is a generic architecture, valid for a particular domain, with no implementation aspects. It is reusable, extendable, and configurable*
- *It is a kind of pattern for whole architectures and it can be instantiated into a specific software architecture by adding implementation-oriented aspects.*

Cloud Architecture Overview



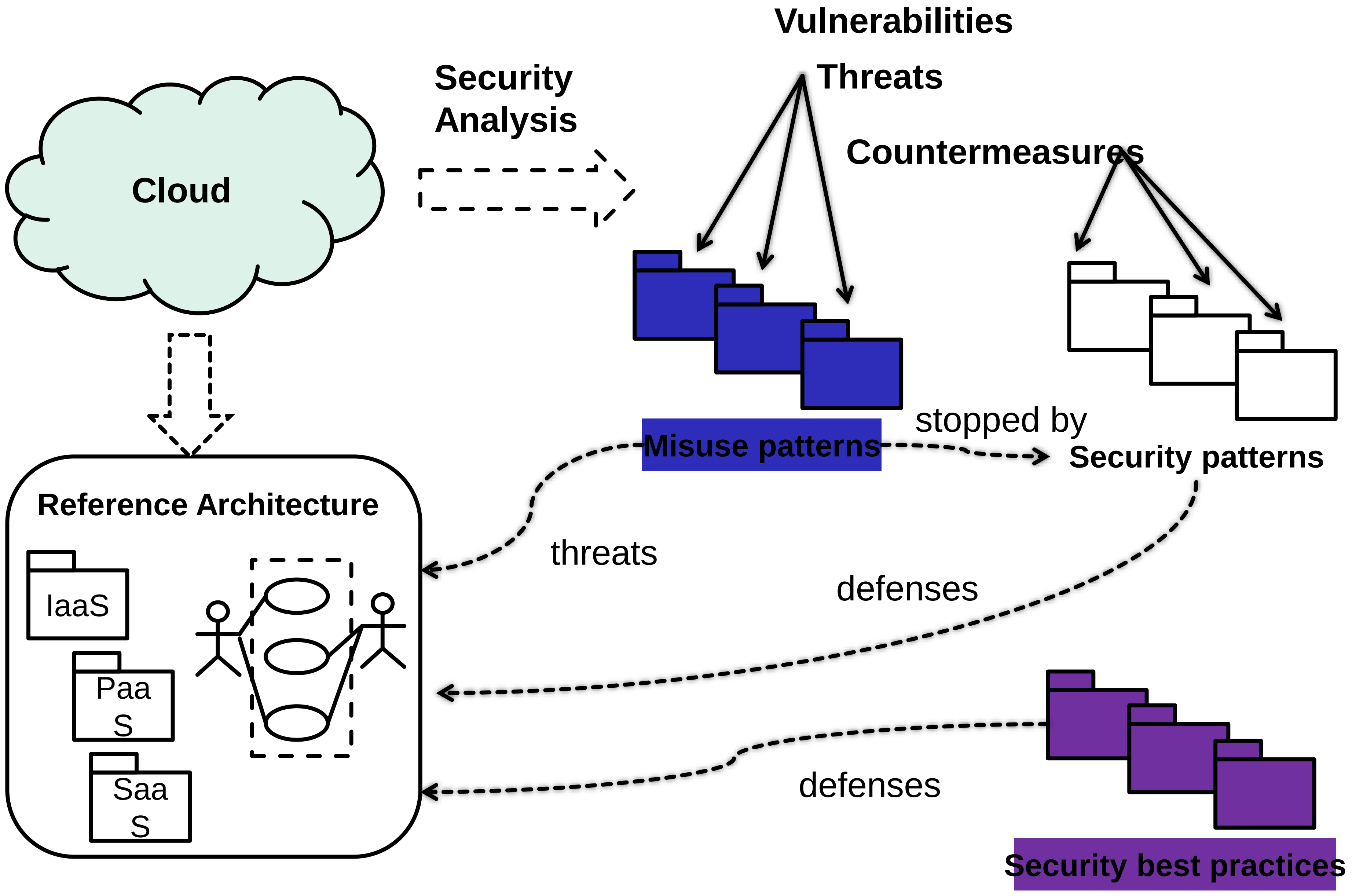
Cloud Architecture Overview



Class Diagram for Infrastructure-as-a-Service architecture

Securing an RA

- ***We start from a list of use cases which describe the typical cloud uses and their associated roles. Lists of cloud use cases are shown in [Bad12], [nis11], and [Has13c].***
- ***We analyze each use case looking for vulnerabilities and threats as in [Bra08]. This implies checking each activity in the activity diagram of the use cases to see how it can be attacked. This approach results in a systematic enumeration of threats.***
- ***We use the list of threats from [Has13a] to confirm these threats and to find possible further vulnerabilities and threats.***
- ***These threats are expressed in the form of misuse patterns. We developed some misuse patterns for Cloud Computing in [Has13b], we show more later in the paper.***
- ***We apply policies to handle the threats and we identify security patterns to realize the policies. There are some defenses that come from best practices and others that handle specific threats. There are also regulatory policies which are realized as security patterns.***
-


Vulnerabilities
Threats
Security Analysis
Countermeasures
Cloud
Misuse patterns
stopped by
Security patterns
Reference Architecture
threats
defenses
IaaS
PaaS
SaaS
defenses
Security best practices

: Threat List vs. Defenses

ID	Threats	Defense
T11	The cloud consumer is malicious and inserts malicious code into the VMI	Authenticator - Authorizer
T21	An external attacker listens to the network to obtain information about the VMI	Secure Channel
T22	VMI may be modified while in transit	Secure Channel
T23	Disavows sending a VMI	Security Logger/Auditor
T31	The IaaS administrator is malicious and collects information within the VMI	Authenticator - Authorizer
T32	The IaaS disavows receiving a VMI	Security Logger/Auditor
T33	Insert malicious code in the image	Authenticator - Authorizer
T41	The IaaS administrator stores a malicious VMI	Authorizer – Authorizer Filter

Cloud security patterns

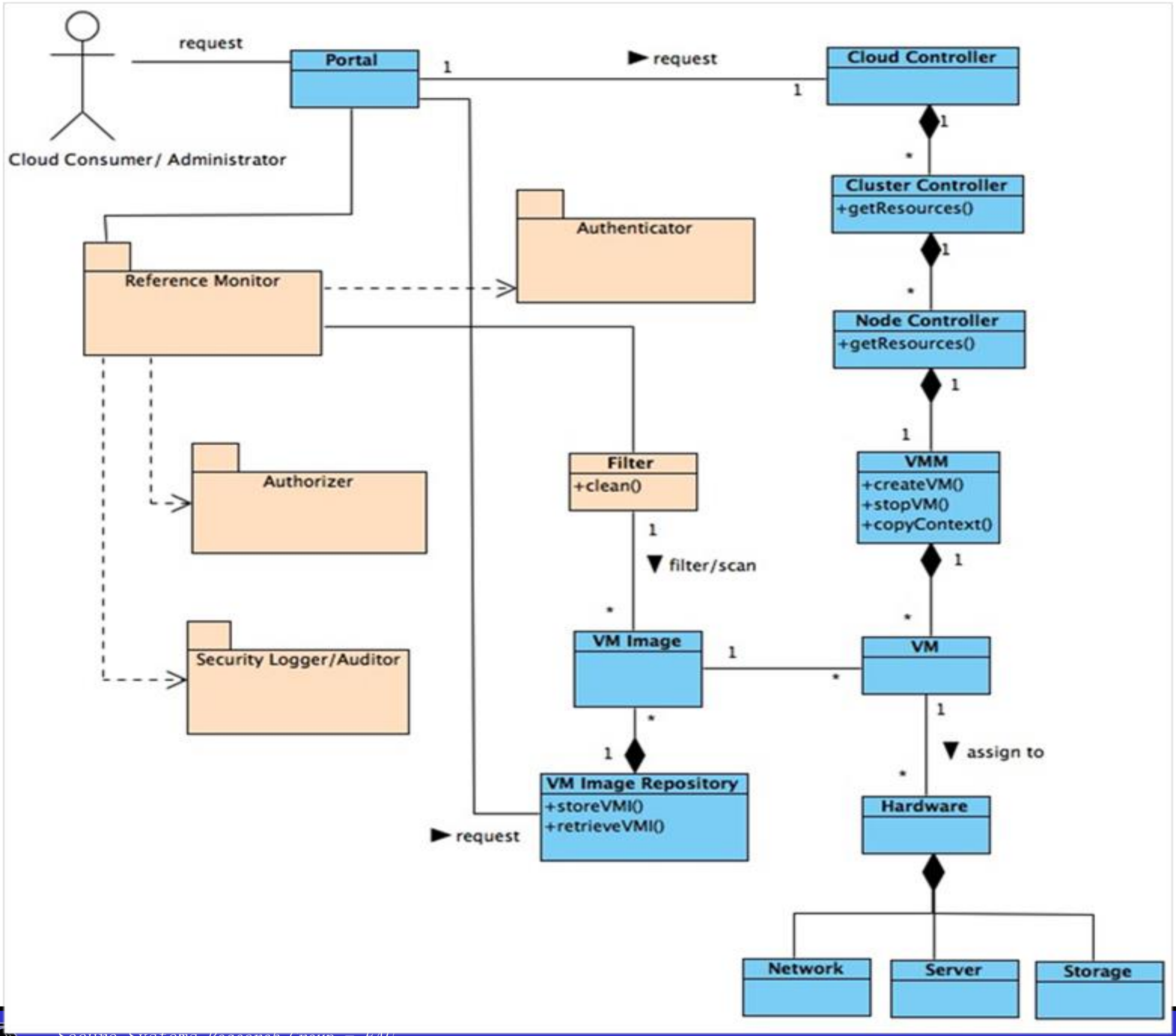
- ***Secure migration process – provides protection for live and offline migration.***
- ***Secure hypervisor – reinforces the security of the hypervisor to avoid some attacks.***
- ***Secure virtual network – secures the communication among virtual machines.***
- ***Virtualized Trusted Platform – provides a framework to determine whether the environment is secure before launching a virtual machine.***
- ***Secure DNS, where Access Control Lists (ACLs) are used to protect the DNS***
- ***Security Group Firewall. A Security Group Firewall divides the firewall in customer groups that have similar filtering requirements [Fer14b]***
- ***Cloud-based Web Application Firewall (CWAF). Controls access to web applications communicating through HTTP according to authorization rules with the objective of stopping XSS, SQL injection, and similar attacks.***

Some misuse patterns for clouds

- ***Covert channels in clouds – covert channels allow inter-VM communication bypassing the security rules of the hypervisor.***
- ***Virtual machine escape – describes how to exploit the hypervisor in order to take control of the underlying platform.***
- ***Virtual machine hopping – describes how a virtual machine can access other virtual machines, for example by exploiting the hypervisor.***
- ***Sniffing virtual networks – describes how a virtual machine can listen to the virtual network traffic in order to get confidential information.***
- ***Spoofing virtual networks – describes how a malicious virtual machine can intercept information in the virtual network with the purpose of altering its routing function.***

Secure Reference Architecture

- ***The identified threats can be neutralized by applying appropriate security patterns.***
- ***Each threat can be controlled by a corresponding security pattern. Once security patterns are identified, we apply them into the reference architecture in order to stop or mitigate the threats***
- ***Security mechanisms are added to the basic RA, including Authenticator, Authorizer, Security Logger/Auditor and others that mitigate specific threats***


Cloud Consumer/ Administrator
request
Portal
1
► request
1
Cloud Controller
1
*
Cluster Controller
+getResources()
1
*
Node Controller
+getResources()
1
1
VMM
+createVM()
+stopVM()
+copyContext()
1
*
VM
1
▼ assign to
*
Hardware
Network
Server
Storage
Authenticator
Reference Monitor
Authorizer
Security Logger/Auditor
Filter
+clean()
1
▼ filter/scan
*
VM Image
1
*
*
1
VM Image Repository
+storeVMI()
+retrieveVMI()
► request

Pattern mining

- *How do we find patterns?*
- *Look at the architecture of systems you know, can you see similar ideas?*
- *Look for analogies in systems. If two or more systems have similar ways of solving a problem you have a pattern*
- *Describe standards and regulations, e.g. HIPAA, Sarbanes-Oxley, FEMA. Web services security standards*

Conclusions

- ***Security patterns are a useful tool to build secure architectures***
- ***They require appropriate methodologies to use them, good catalogs and tools***
- ***They provide a good way to handle security in a holistic way, necessary for complex systems***
- ***Patterns are also valuable for evaluating existing systems and for teaching security concepts***
- ***Reference architectures can simplify secure application development or can be used to build secure architectures that conform to some type of application, e.g. clouds***

Conclusions II

- ***Patterns cannot prevent attacks that happen through code flaws but can make their effect much less harmful***
- ***Patterns can be made more formal: OCL***
- ***Security patterns are now accepted by many companies, Microsoft, Sun, Siemens, and IBM have books, papers, and web pages on this subject.***
- ***A general page for security patterns: www.securitypatterns.org***

Papers

Look in my web page:

http://www.cse.fau.edu/~ed

Write to: ***ed@cse.fau.edu***

A larger set of slides is in: http://faculty.eng.fau.edu/fernande/2015/08/24/re15-slides/