Security patterns and secure systems design using UML

Eduardo B. Fernandez
Dept. of Computer Science and Eng.
Florida Atlantic University
www.cse.fau.edu/~ed

About me

- Professor of Computer Science at Florida Atlantic University, Boca Raton, FL., USA
- Worked 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,…

About this tutorial

- Its objective is to provide an approach for designing secure systems using patterns
- Emphasis on important new developments: web services, standards, patterns, UML
- An engineering, not theoretical, approach.

Objectives

- To apply a systematic approach to secure systems development
- To get a panorama of security patterns and how to use them
- To be able to evaluate the security of a system (in a qualitative way)
- To get ideas for research

Outline I

- Security concepts and definitions
- The Internet
- Attacks
- The design of secure systems
- Security models
- Firewalls
- Operating systems

Outline II

- Web services
- Application security
- Distributed and web systems
- Security patterns
- Applying the patterns
- Conclusions

Security

- Objectives
- Countermeasures
- Security architecture
- An example

Value of information

- We rely on information for our credit, health, professional work, business, education
- Illegal access (reading or modification) to information can produce serious problems

Security objectives

- Confidentiality--no leakage of sensitive or private information
- Integrity-- no unauthorized modification or destruction of information
- Availability (No denial of service) -- annoying , costly
- Lack of accountability (Non-repudiation)-- legally significant

The meaning of security

- Security implies providing these objectives in the presence of attacks
- Security requires technical, managerial, and physical countermeasures (defenses)
- We only consider technical aspects here
- A related aspect is privacy, a legal and ethics concern

Countermeasures

- Identification and Authentication– first step
- Access control/ authorization --provide confidentiality and integrity
- Auditing-- basis for prosecution or improvements to the system
- Cryptography-- a mechanism to hide information and prove identity and rights
- Intrusion detection

Security architecture

- *Authorization rules* define what is allowed or not allowed (who can see what and how)
- The lower levels must enforce these rules
- *Assurance* is a measure of how well the lower levels enforce the rules

The Internet

- Basic architecture
- Documents
- New architectures

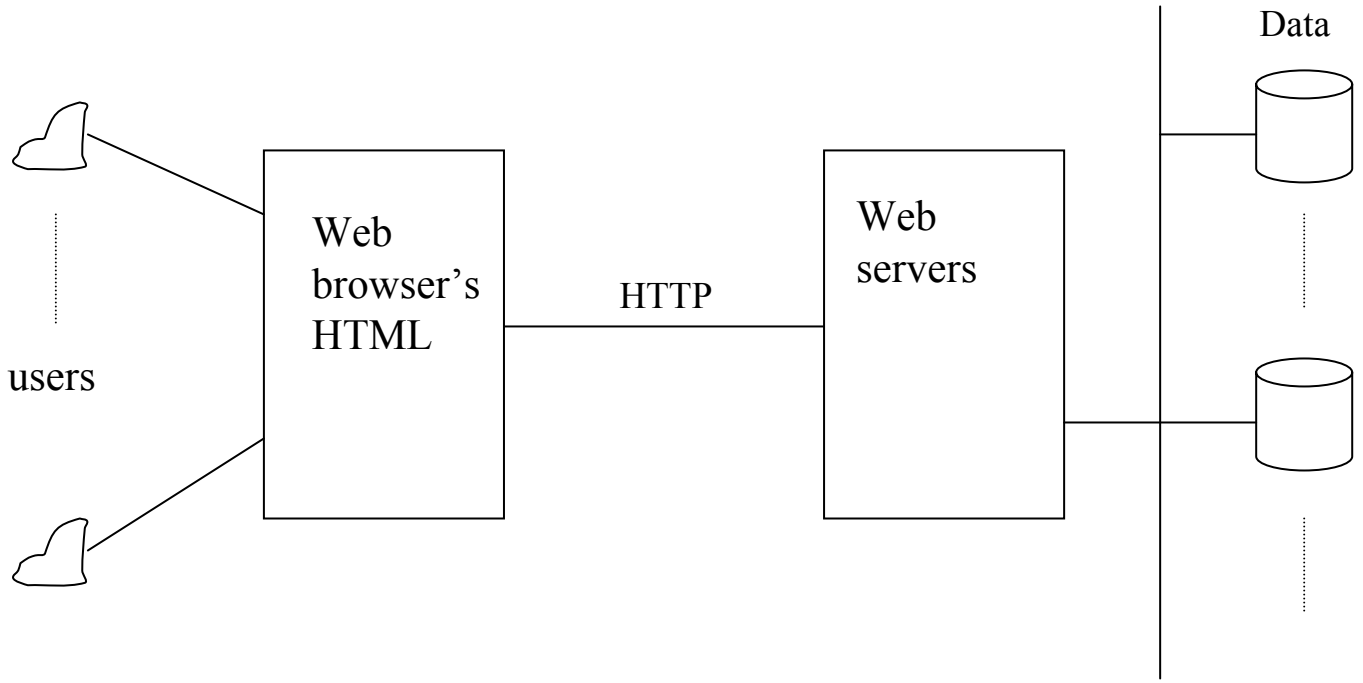
Architectures

- The architecture of a system defines the system in terms of components (units) and of interactions between these units. Architecture includes: system topology and organization, decomposition into components, assignment of functionality to components, component interactions, system properties (nonfunctional requirements, e.g. performance, security), correspondence between requirements and units.

Basic Architectural components

- Web browsers -- can request HTML documents, provide URL caching , support directories
- Web servers -- receive user requests , find and return documents
- Files or DBMS store documents
- Documents -- pages or sets of pages

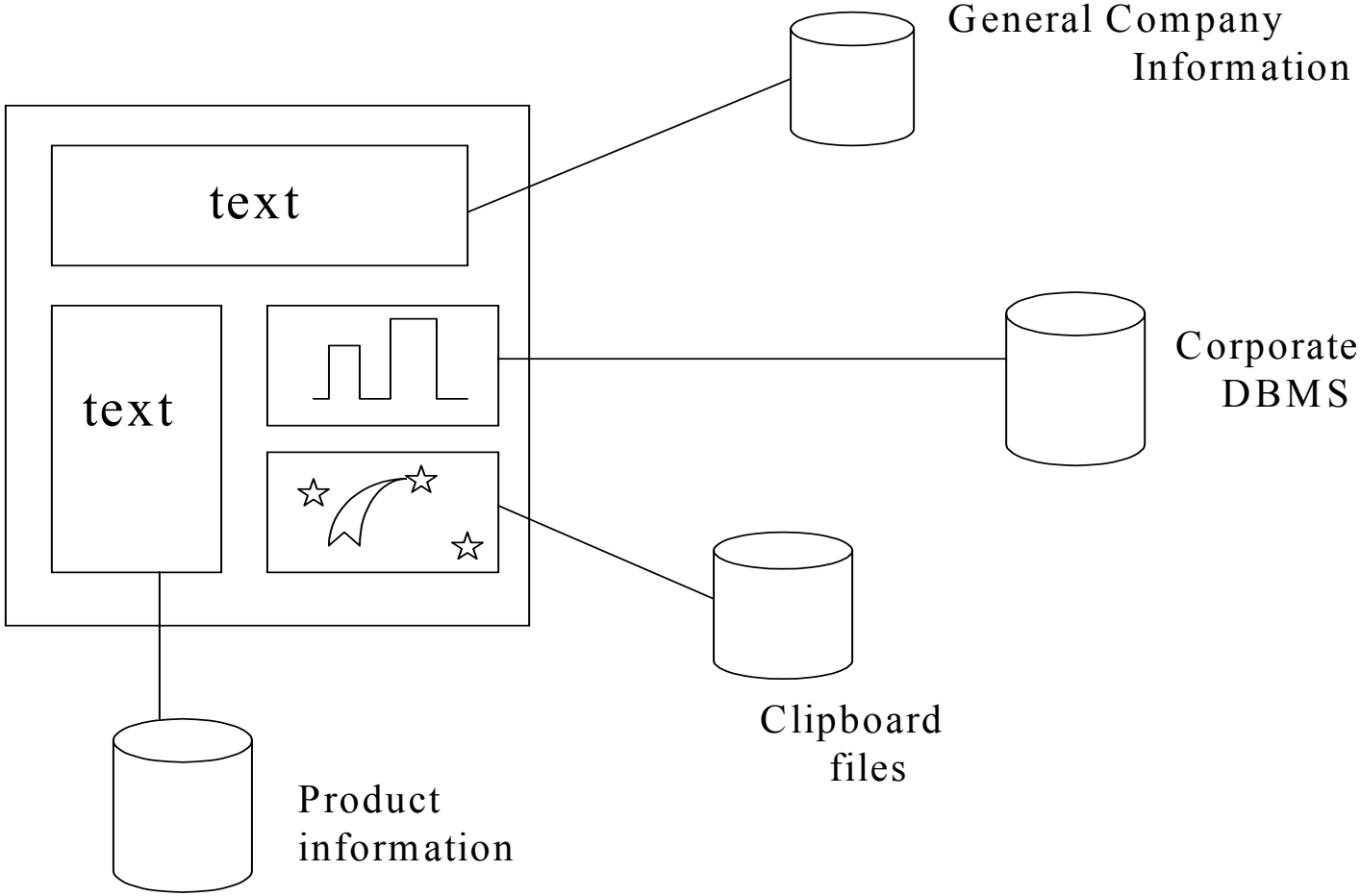
Basic Internet architecture



Web documents

- Hypertext /multimedia
- Passive or active (contain links to programs)
- Fixed or dynamic (assembled on request)
- Potentially all institution data can be considered documents

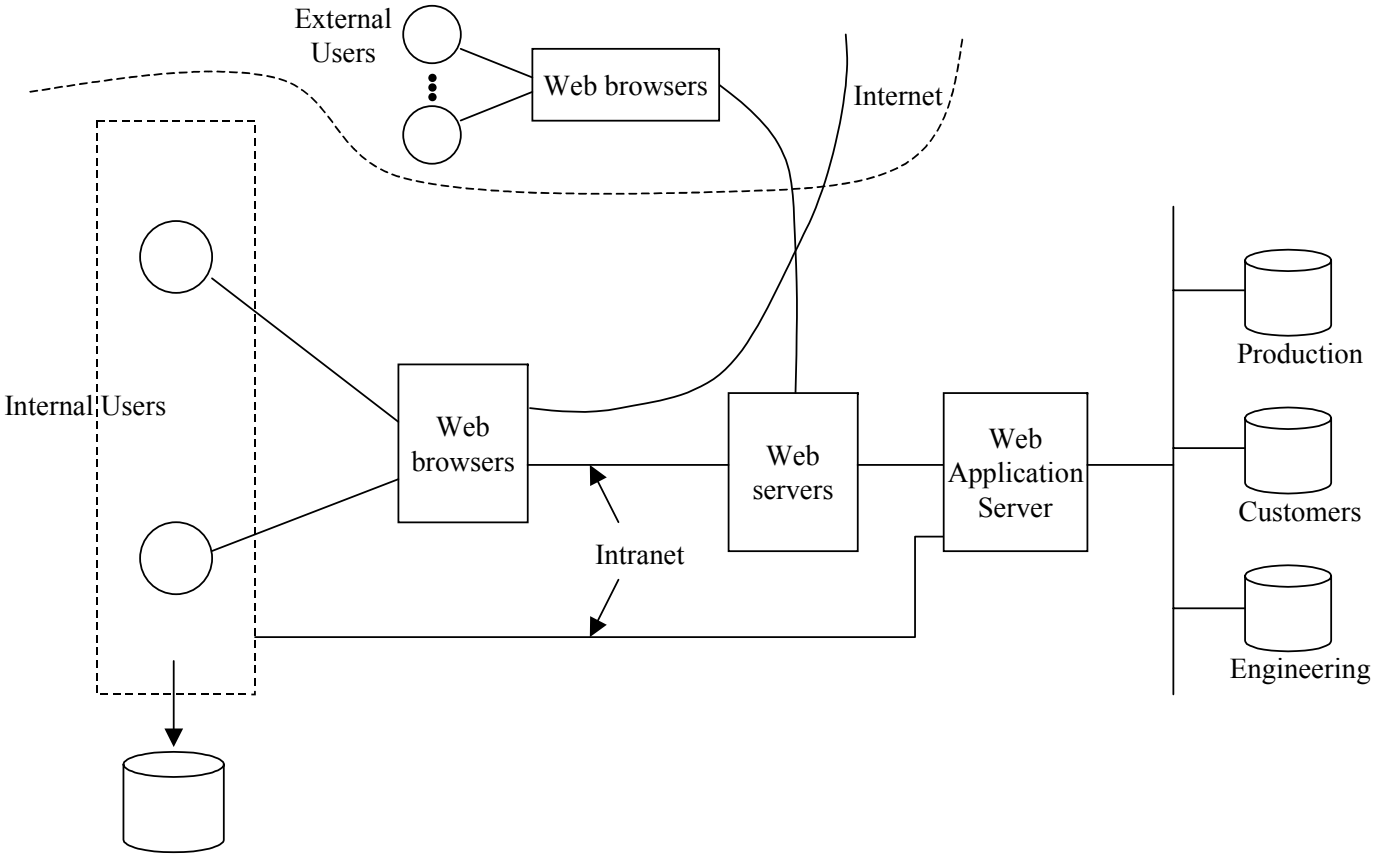
Example of a web page



XML

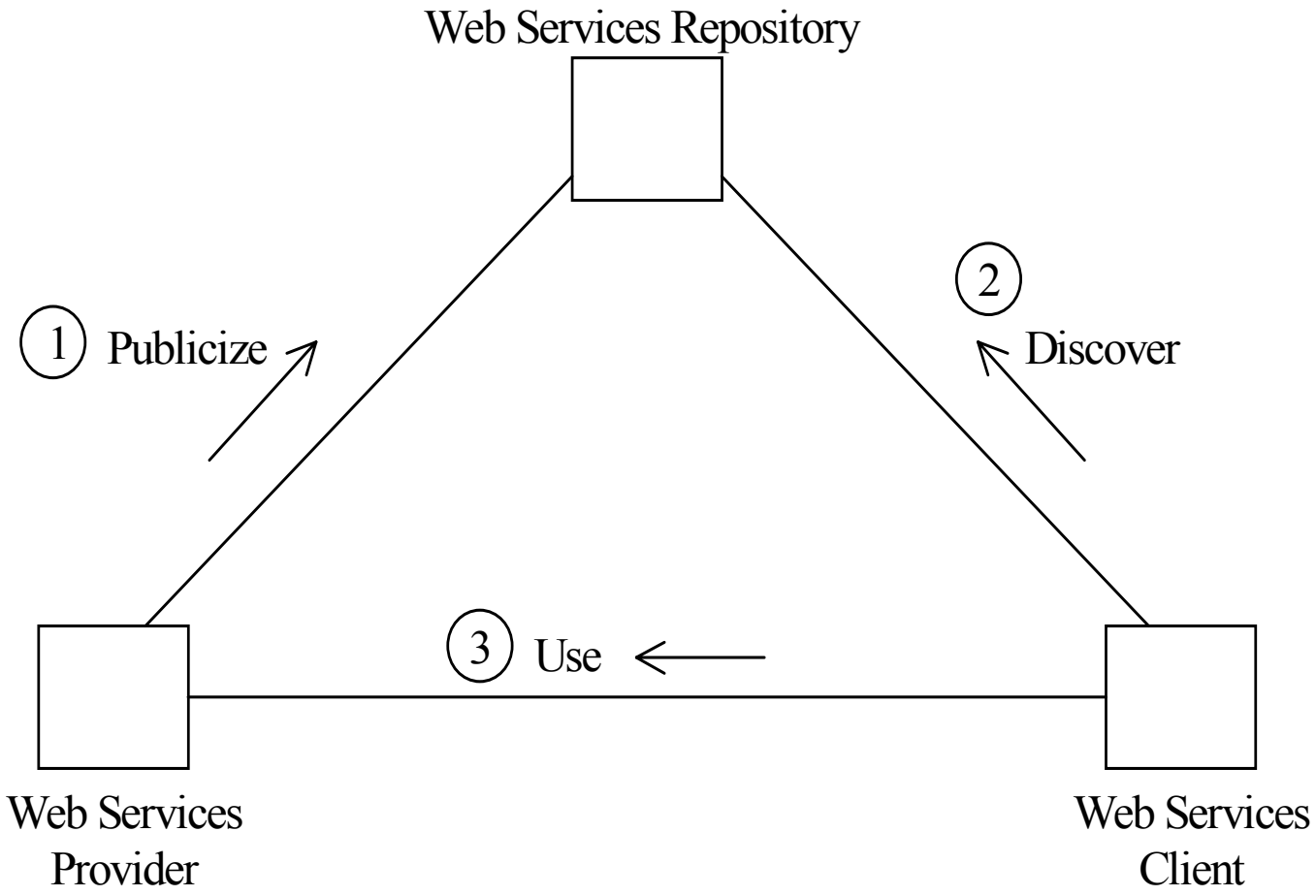
- XML is a metalanguage to define the meaning and structure of documents. A subset of SGML (Standard Generalized Markup Language). Basic ideas: use tags in data items to define their meaning, relate data items through nesting and references.

Enterprise architectures



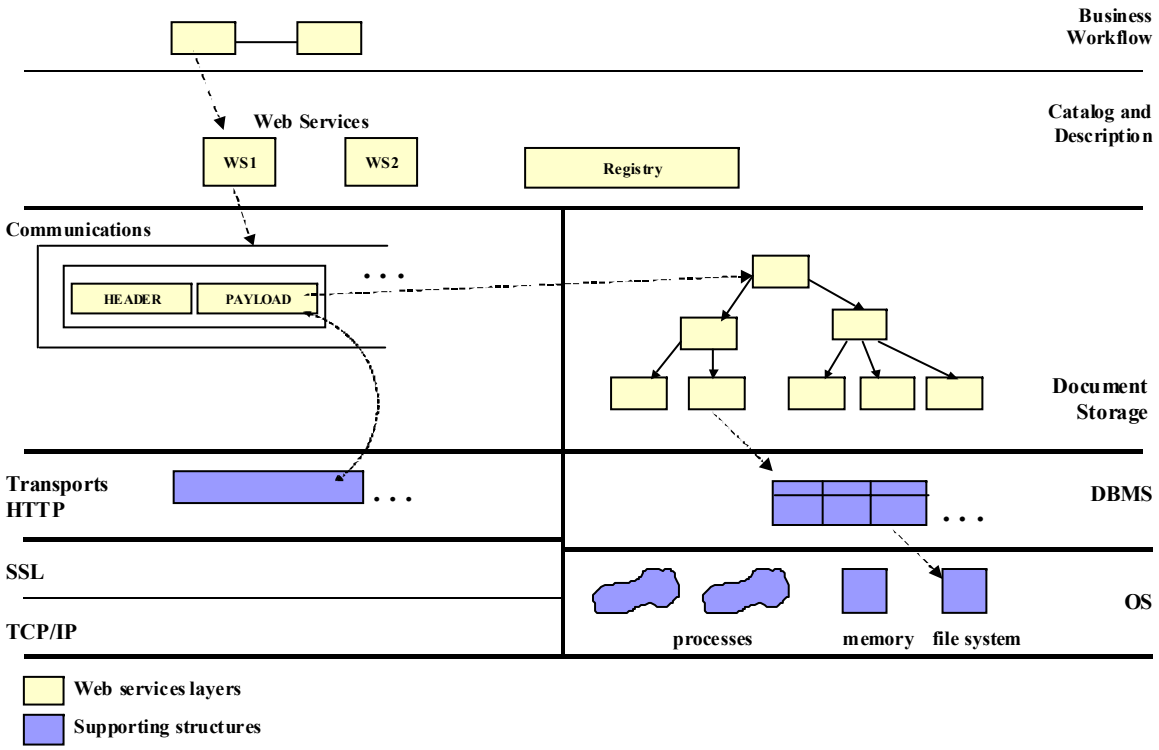
Web Services

- A Web Service is a type of component that is available on the web and can be incorporated in applications or used as a standalone service
- Require a standard supporting framework
- The web is becoming a marketplace of web services


Web Services Repository
1
Publicize
2
Discover
3
Use
Web Services
Provider
Web Services
Client

Web services architectures

- Web services (eServices) are a part of the application layer
- Web services are built out of XML, a lower-level data layer
- A SOAP layer is used for XML message transmission
- Internet layers and web server layers provide support for these layers

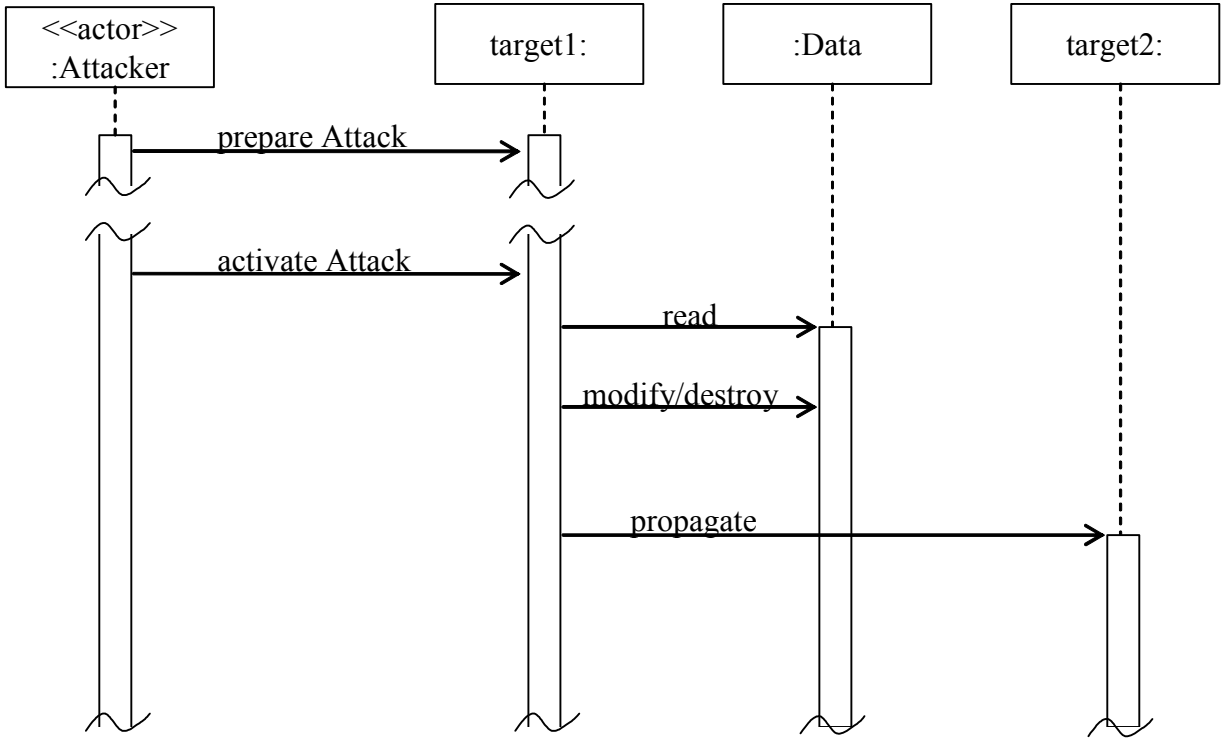

Business Workflow
Web Services
WS1
WS2
Registry
Catalog and Description
Communications
HEADER
PAYLOAD
Document Storage
Transports HTTP
DBMS
SSL
TCP/IP
processes
memory
file system
OS
Web services layers
Supporting structures

Attacks

- Methods
- Types

Attack methods

- Preparation—Information gathering, scanning, planting malicious code, masquerading (spoofing)
- Activation—perpetrator-controlled, timed, victim activated
- Mission—active (affects integrity and availability), and passive misuse (eavesdropping, inference), denial of service


<<actor>>
:Attacker
target1:
:Data
target2:
prepare Attack
activate Attack
read
modify/destroy
propagate

Malicious code

- *Trojan Horses*—A Trojan Horse is an apparently useful program that has harmful hidden functions
- *Viruses* – A virus is a program that attaches itself to another program, propagates, and usually causes some data destruction.
- *Worms*—A worm is a program that propagates itself without infecting the host.

More varieties

- Spyware—collect passwords, credit card numbers, or general info. (adware)
- Spam—wholesale sending of messages, can be used to propagate viruses
- Phishing messages—enticing users to disclose information

Current situation

- The Internet is an insecure place and attacks keep occuring
- One of the main reasons is the poor quality of the software used in operating systems and applications
- We need a systematic way to build secure software

The design of secure systems

- Security is a nonfunctional aspect that must be satisfied in addition to functional aspects
- We cannot show absence of security flaws
- We must use good development methods and hope for the best
- Add-on security is not the way

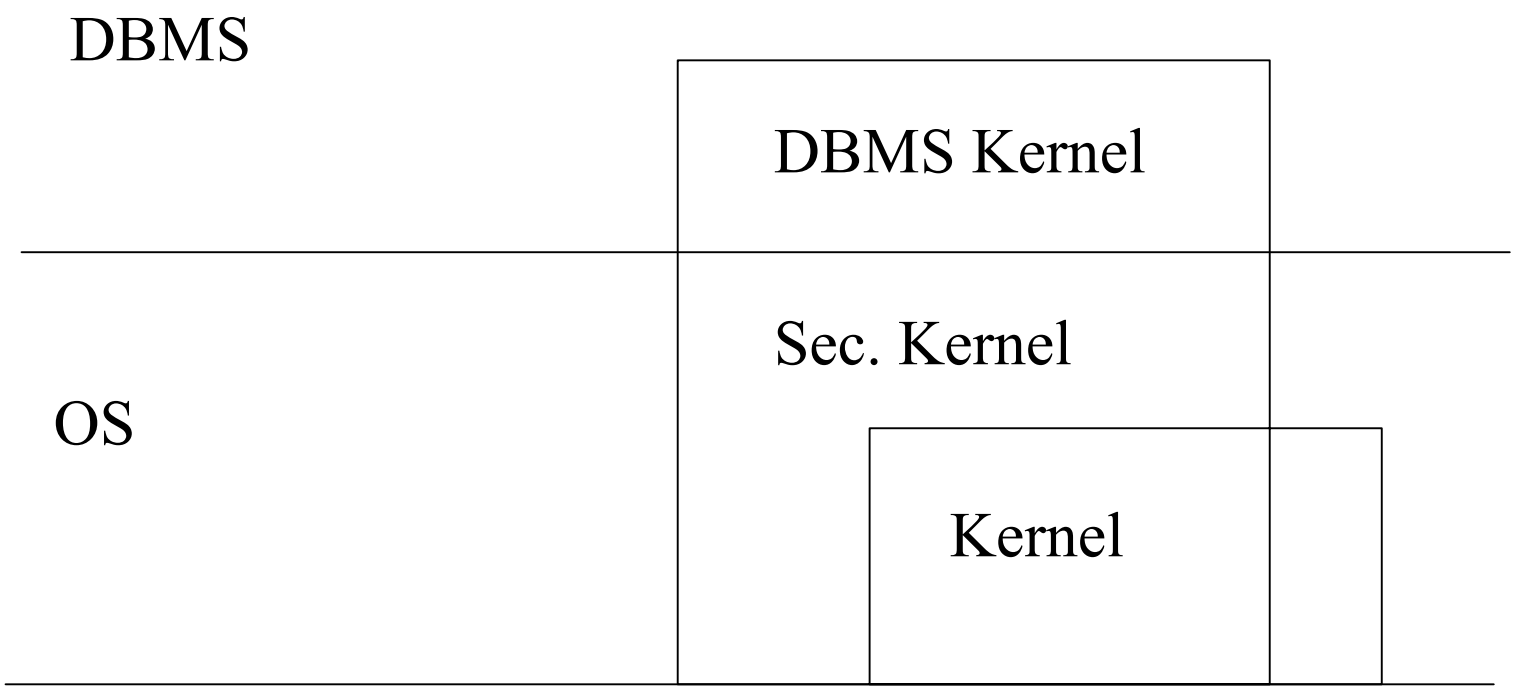
Principles of design for security

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism

Attempted approach

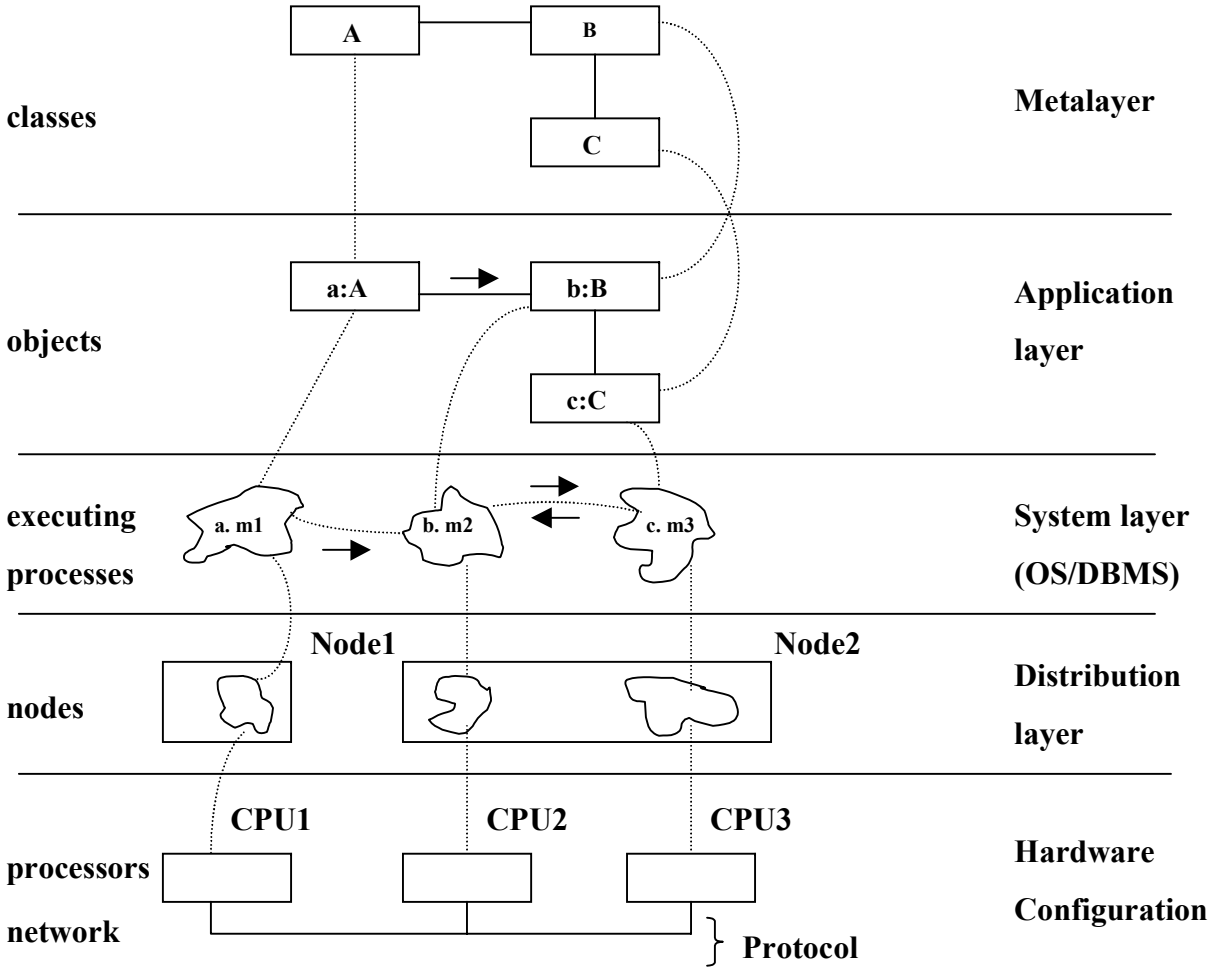
- Define a security kernel: includes all security-related functions
- Verify kernel: possible only for relatively simple systems
- Requires special languages and special operating systems
- Not practical for general systems

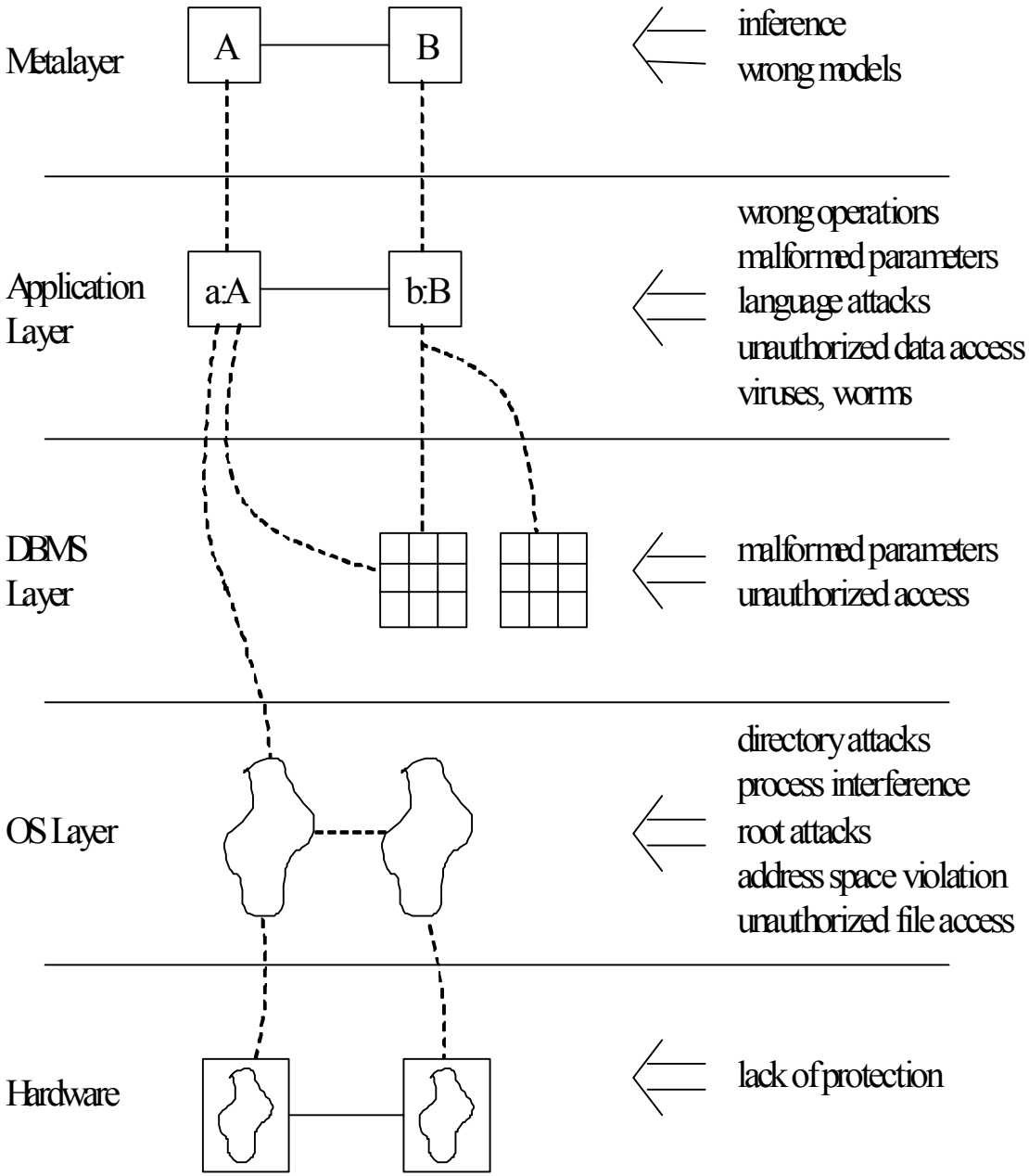
Use of kernels



Applying the principles

- Security should be applied where the application semantics is understood
- Security is an all-levels problem
- We should start from high-level policies and map them to the lower levels
- We need precise models to guide system development

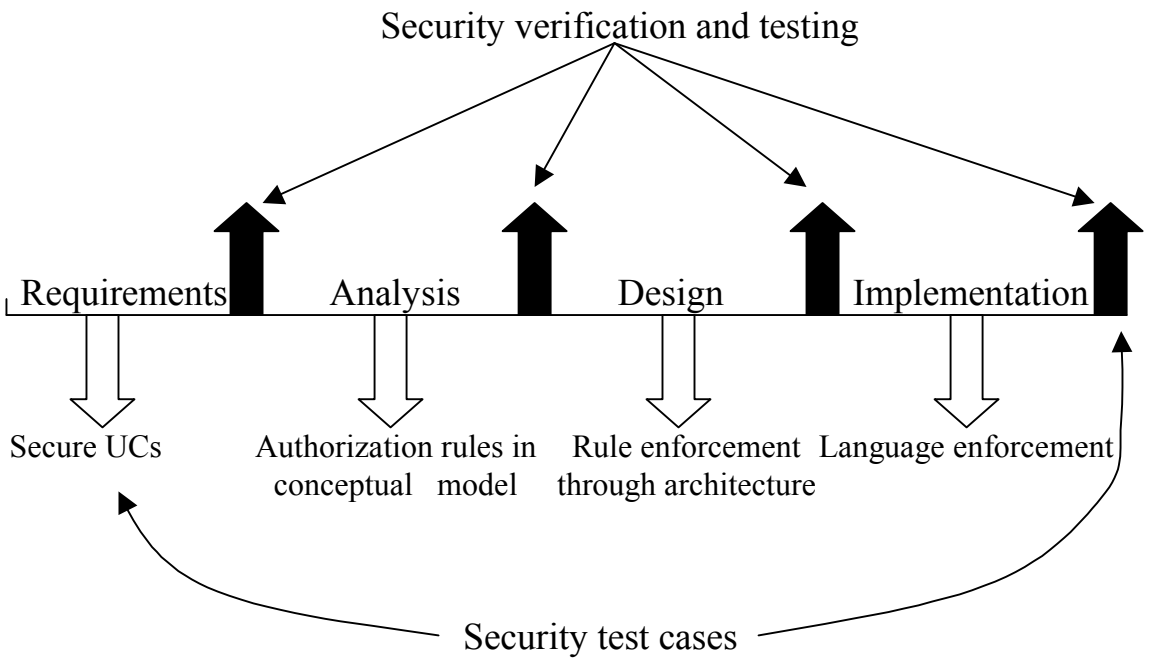

A
B
C
classes
Metalayer
a:A
b:B
c:C
objects
Application
layer
executing
processes
a. m1
b. m2
c. m3
System layer
(OS/DBMS)
Node1
Node2
nodes
Distribution
layer
CPU1
CPU2
CPU3
processors
network
Protocol
Hardware
Configuration


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
lack of protection

Secure systems development methodology

- Apply security principles throughout the whole software lifecycle
- Use of object-oriented design
- Use cases define rights for roles
- Patterns build a secure conceptual model
- Multilayer architecture extends the model to the lower architectural levels

Software lifecycle



Use of object-oriented modeling

- Strong conceptual modeling capability , applicable to hardware, software, applications, authorization rules
- Abstraction from irrelevant details
- Intuitive , graphic, semiformal approach
- Can be enhanced with formal specifications

OO and UML

- UML is an object-oriented language for specifying, constructing, visualizing, and documenting a software design.
- Basically a notation and its corresponding meaning , not a process.
- Combines OMT, Booch, and other ideas.
- OMG standard (www.omg.org)

Use of patterns

- A pattern is a recurring combination of meaningful units that occurs in some context
- Patterns embody experience and good design practices
- Prevent errors, save time
- A good catalog of patterns is needed

Security patterns

- Analysis and design patterns are well established
- There are many principles of good design that have been developed to build secure systems
- We have combined these two ideas showing that it is possible to develop a collection of patterns that can be used to build secure systems
- Now building a catalog of security patterns
- Security patterns page: www.securitypatterns.org

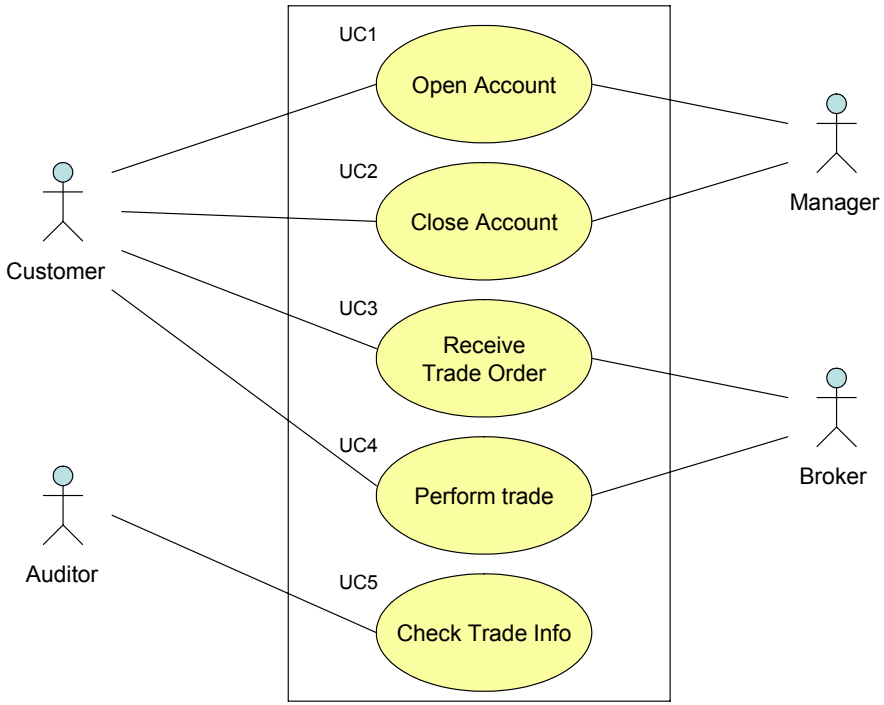
Use patterns at all levels

- 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

Start from requirements

- Use cases define interactions with the system (Use case diagram)
- Use cases include several actions
- We look at the possible attacks to each action
- We can define needed rights for actors to perform the use cases

A financial institution



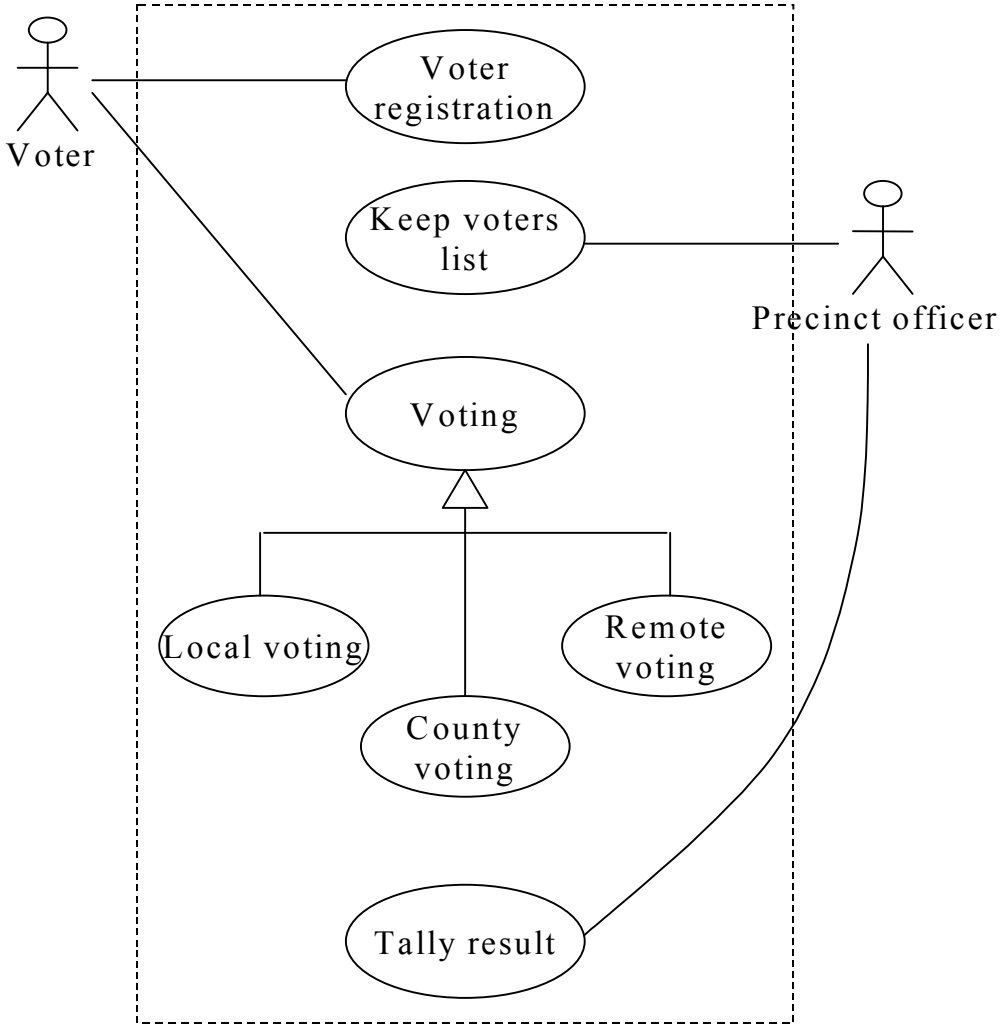
Attacks are related to use cases

- 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.

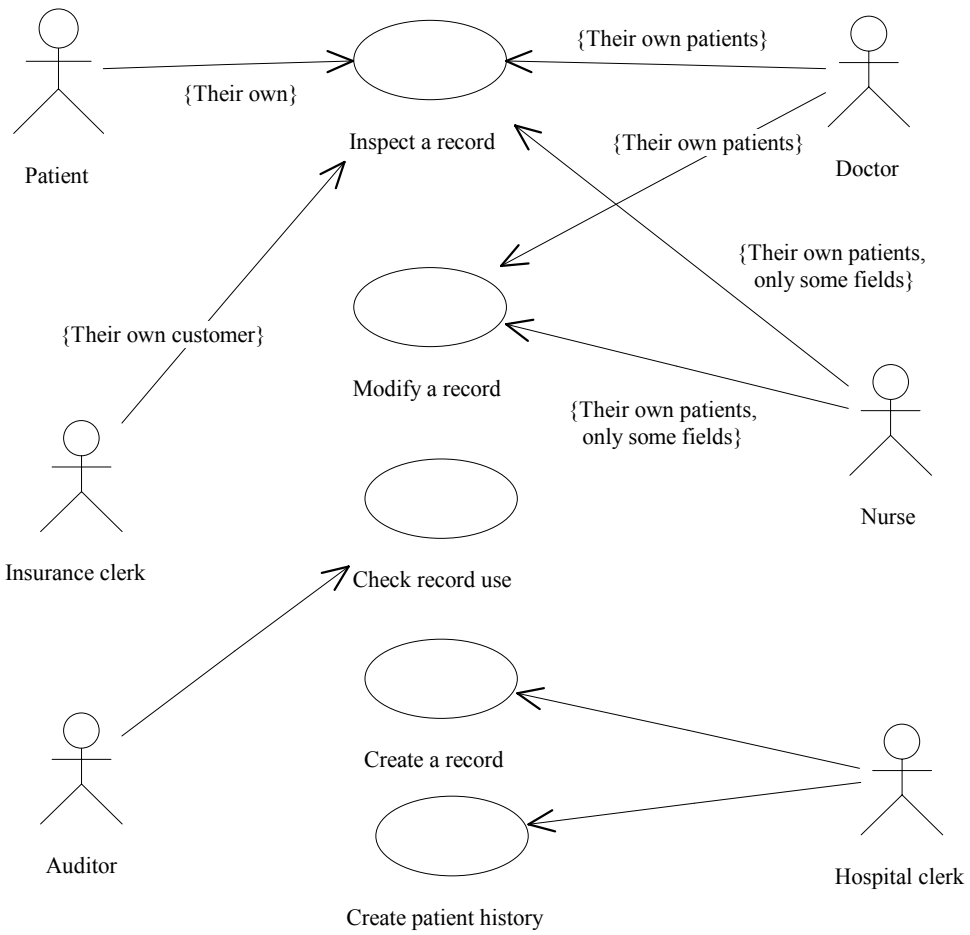
  Stockbroker embezzling customer's money.
- Use Case 5. Impersonation of auditor. Confidentiality violation.


Voter
Voter registration
Keep voters list
Precinct officer
Voting
Local voting
Remote voting
County voting
Tally result

(3) Security model:

Access matrix (mandatory)→RBAC

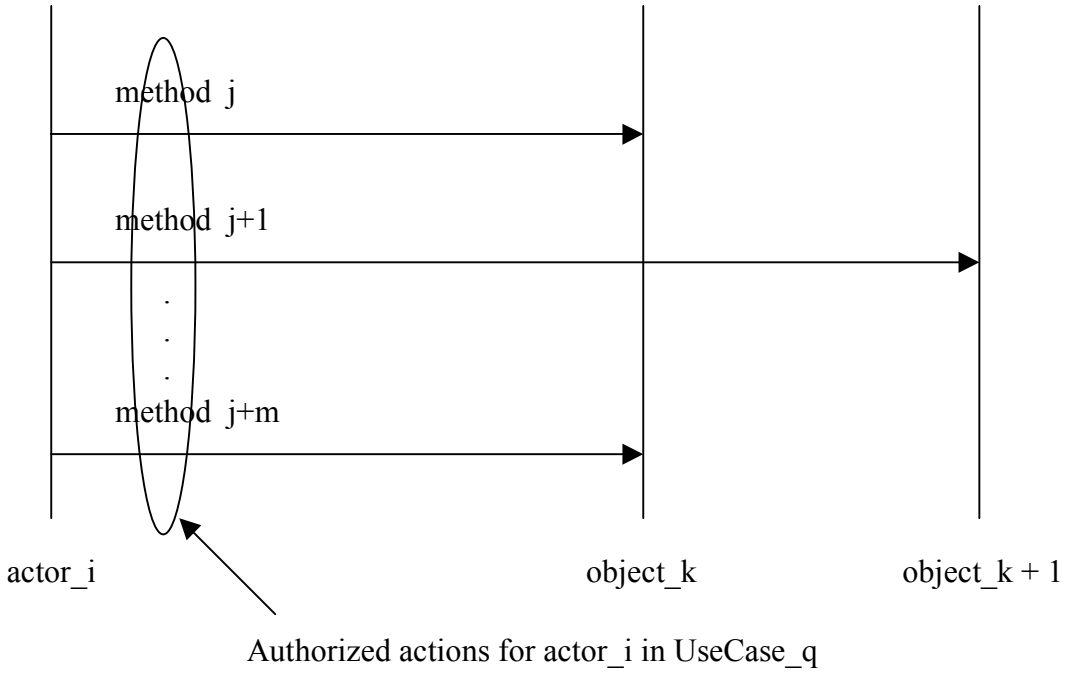
UC:



Use cases to find role rights

- Application of Role-Based Access Control
- Use cases describe all possible uses of the system
- All use cases define all possible and legal accesses
- Each actor role can be given its needed rights

Scenarios to determine rights



Use cases as starting point

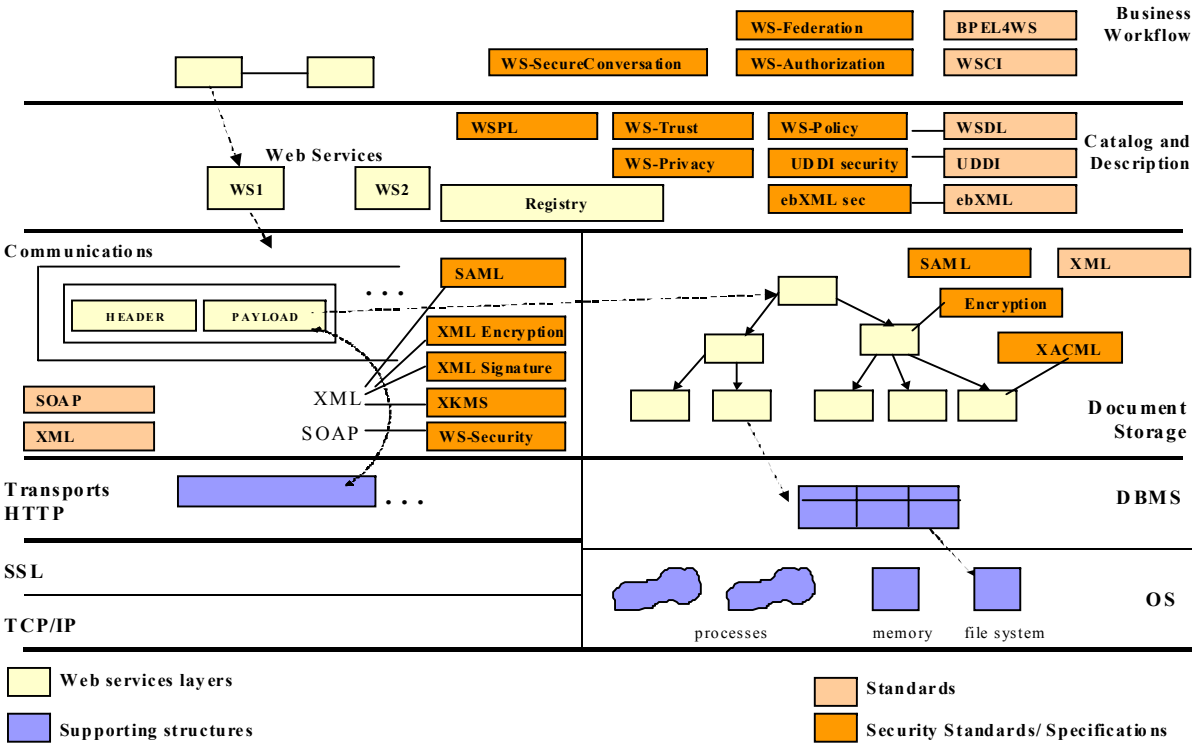
- Attacker is not interested in changing a few bits or destroying a message
- Attacker wants to accomplish some objective, e.g., steal money, steal identity
- This is applying the principle of defining security at the semantic levels
- We also need to comply with standards

Standards

- Orange Book
- Common Criteria (NIST)
- IEEE
- IETF (Internet Engineering Task Force)
- OASIS (Open Applications…)
- W3C
- Industry ad hoc groups: IBM, Microsoft,…

Standards for web services

- A variety of standards to cover all levels
- May overlap or be in conflict
- XACML, WS-Security, SAML, SOAP security, privacy standards
- Confusing for vendors and users


Business Workflow
WS-Federation
BPEL4WS
WS-SecureConversation
WS-Authorization
WSCI
Web Services
WS1
WS2
WSPL
WS-Trust
WS-Policy
WSDL
WS-Privacy
UDDI security
UDDI
Registry
ebXML sec
ebXML
Catalog and Description
Communications
HEADER
PAYLOAD
SAML
XML Encryption
XML Signature
XKMS
WS-Security
SOAP
XML
SOAP
XML
SAML
XML
Encryption
XACML
Document Storage
Transports
HTTP
DBMS
SSL
TCP/IP
OS
processes
memory
file system
Web services layers
Supporting structures
Standards
Security Standards/ Specifications

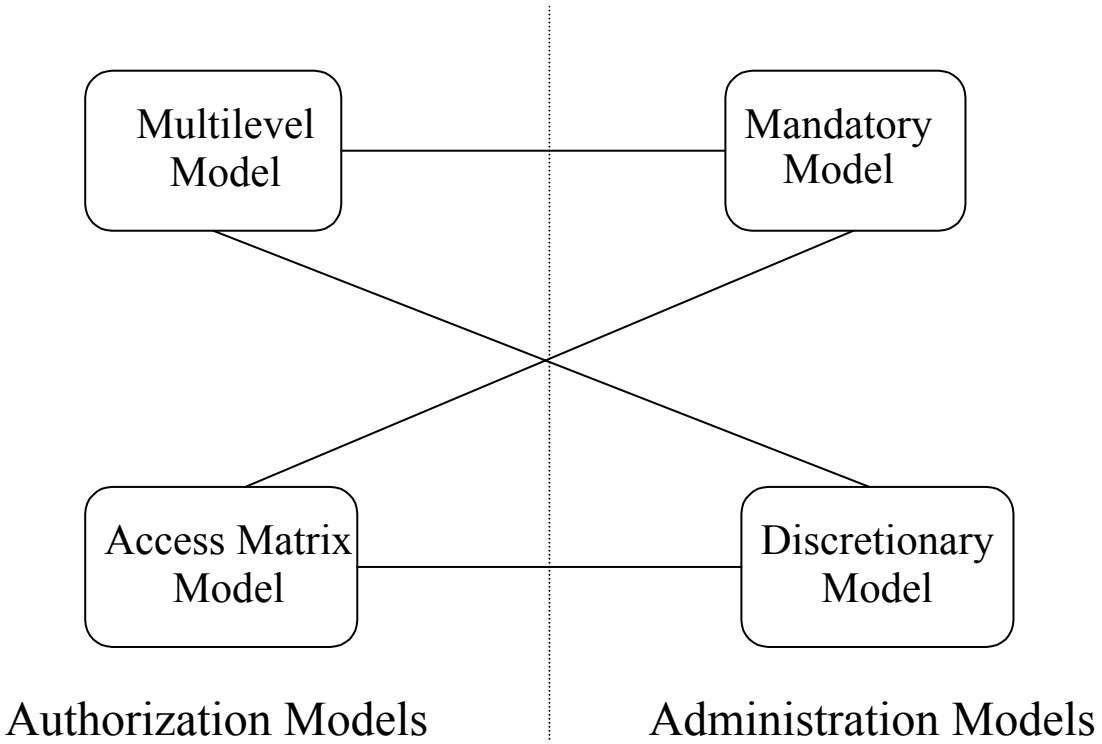
Security models

- Classification
- Access matrix
- Role-Based Access Control
- Multilevel security

Classification of security models

- Multilevel --users and data are assigned security levels
- Access matrix -- subject has specific type of access to data objects
- Mandatory --access rules defined only by administrators
- Discretionary -- users own data and can grant access to other users

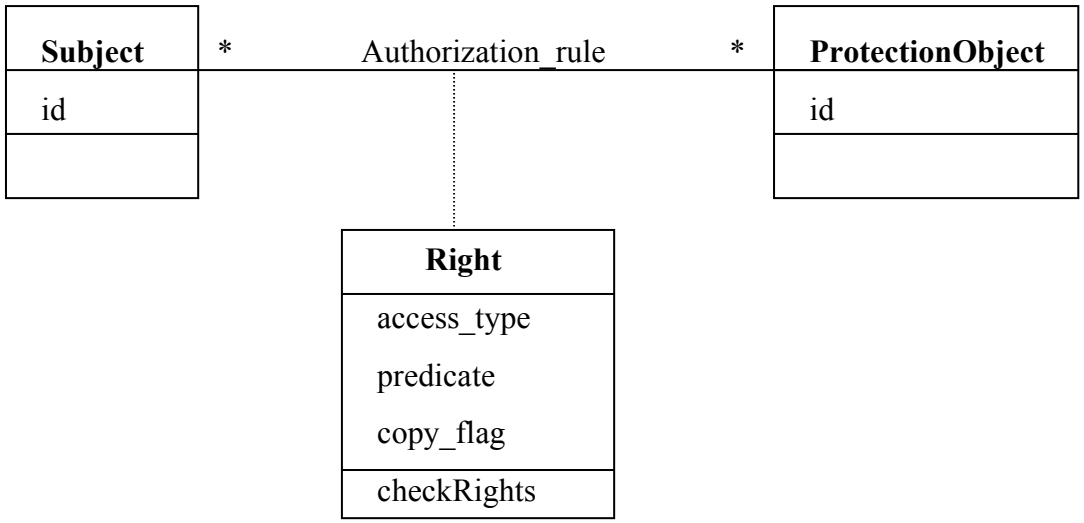
Security models



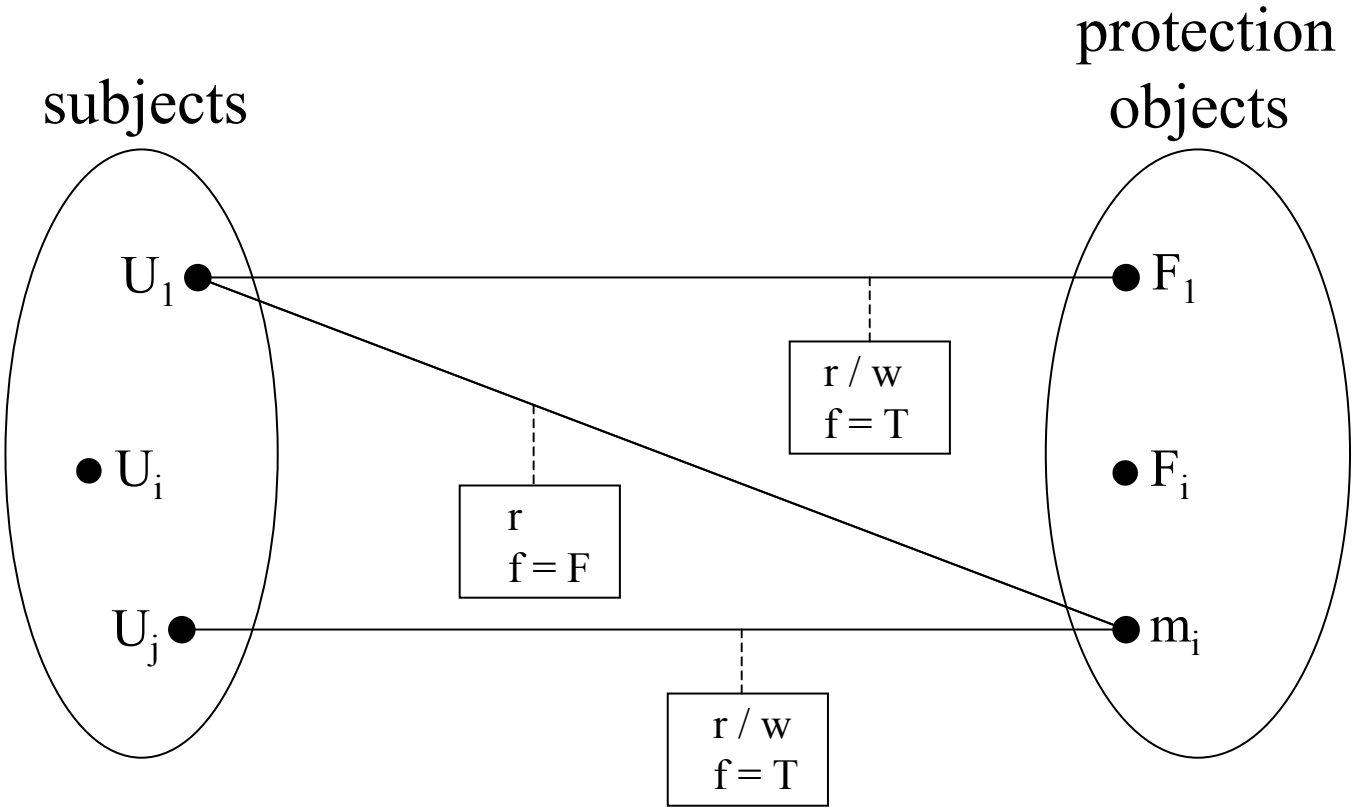
Access matrix authorization rules

- Basic rule (s, o, t) , where s is a subject (active entity), t is an access type, and o is an object
- Extended rule (s, o , t , p, f) , where p is a predicate (access condition or guard) and f is a copy flag
- This, and the other models, can be described by OO patterns

Authorization pattern



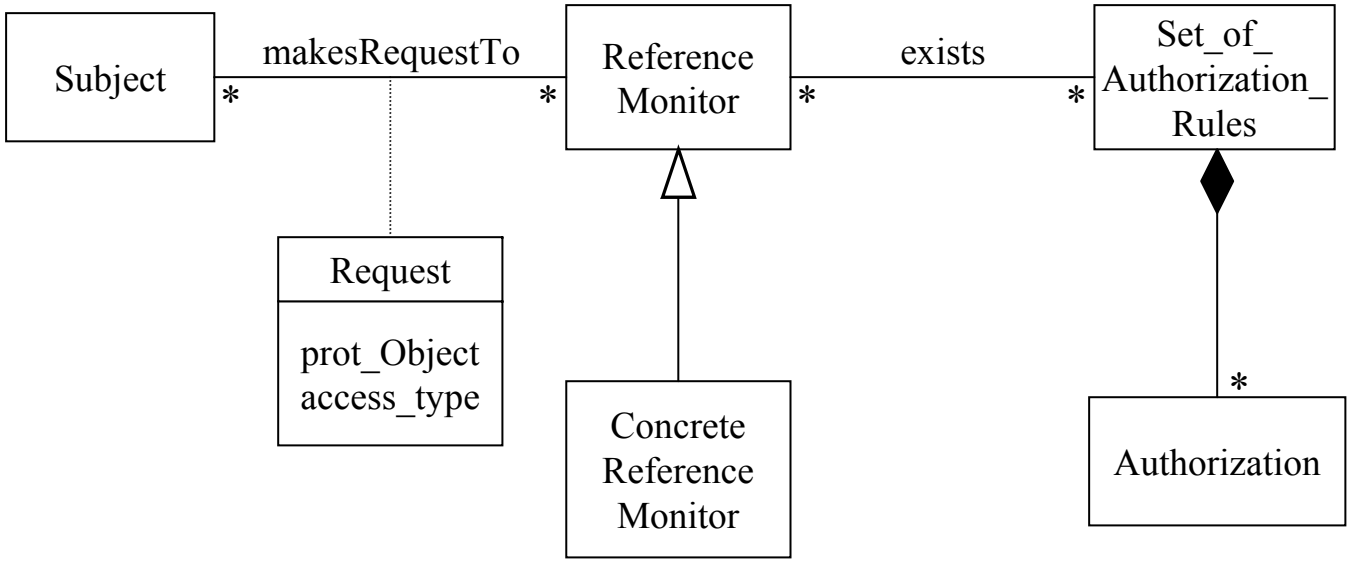
Authorization mapping



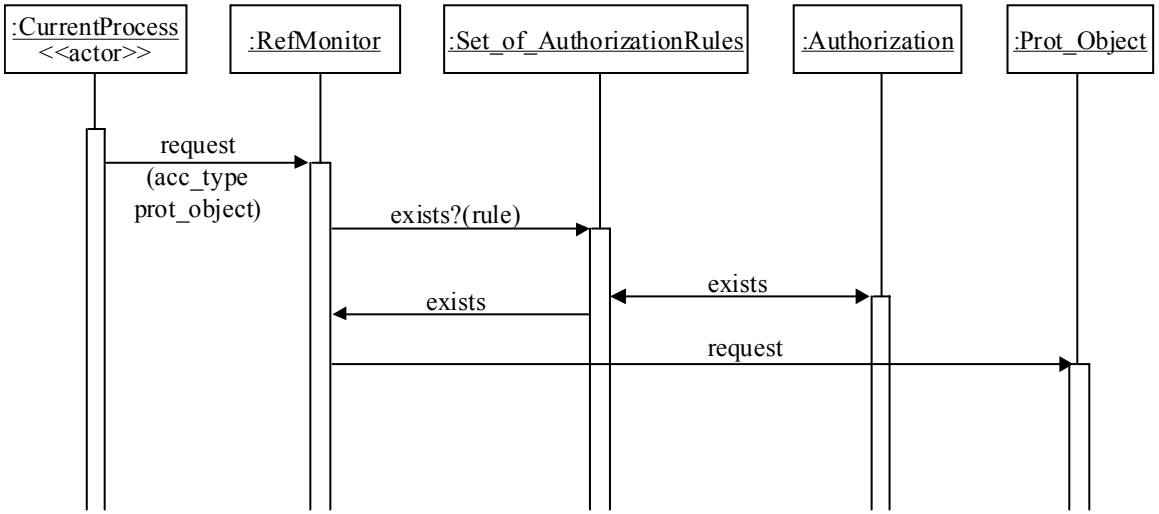
Reference Monitor

- 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 pattern



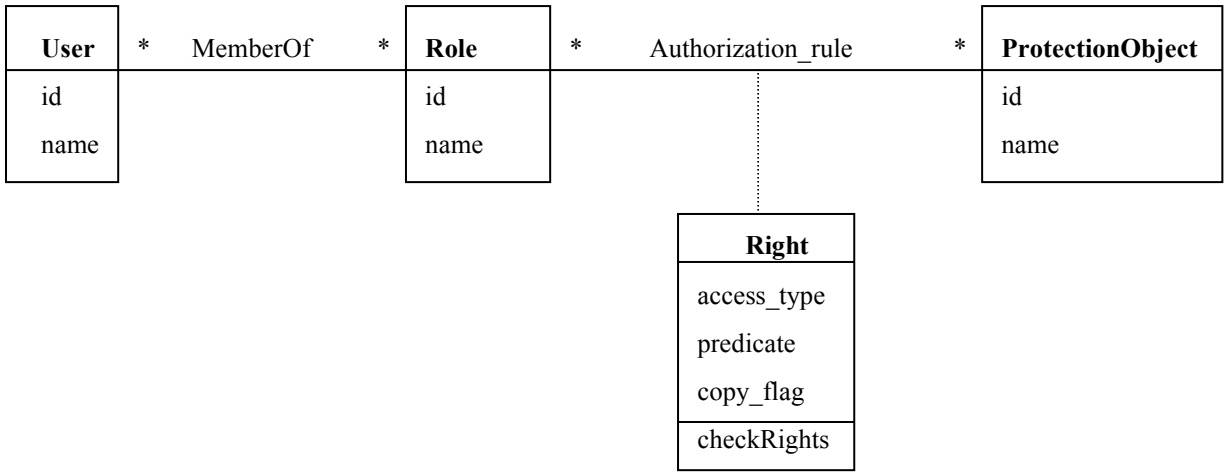
Enforcing access control



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

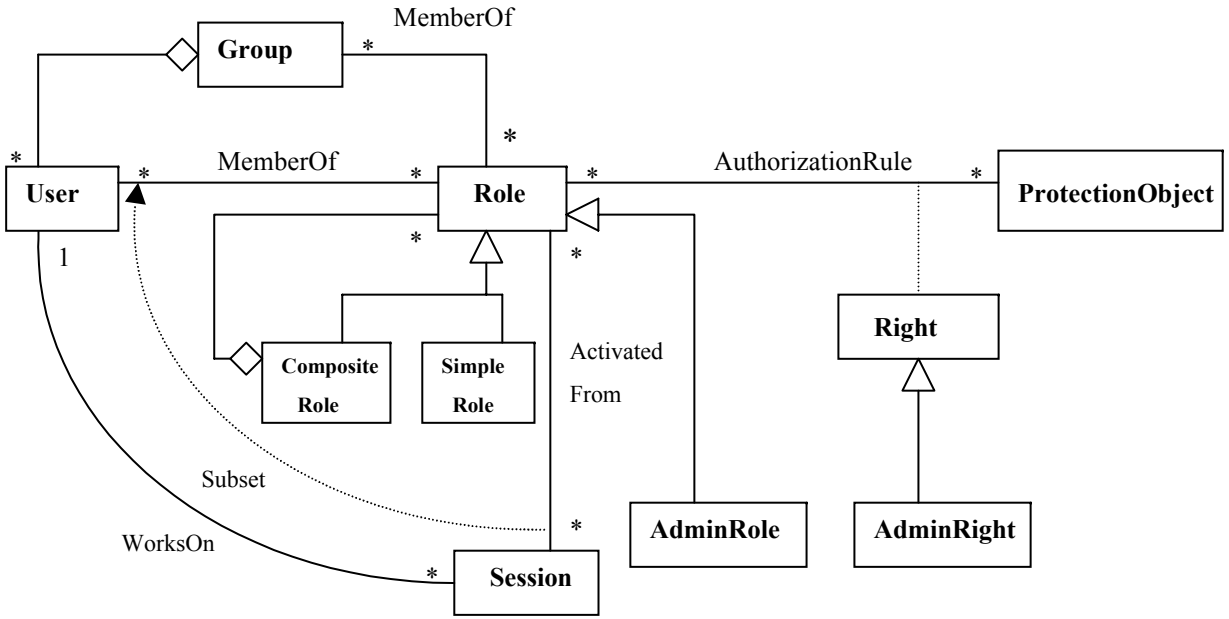
Basic RBAC pattern



Extended RBAC

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

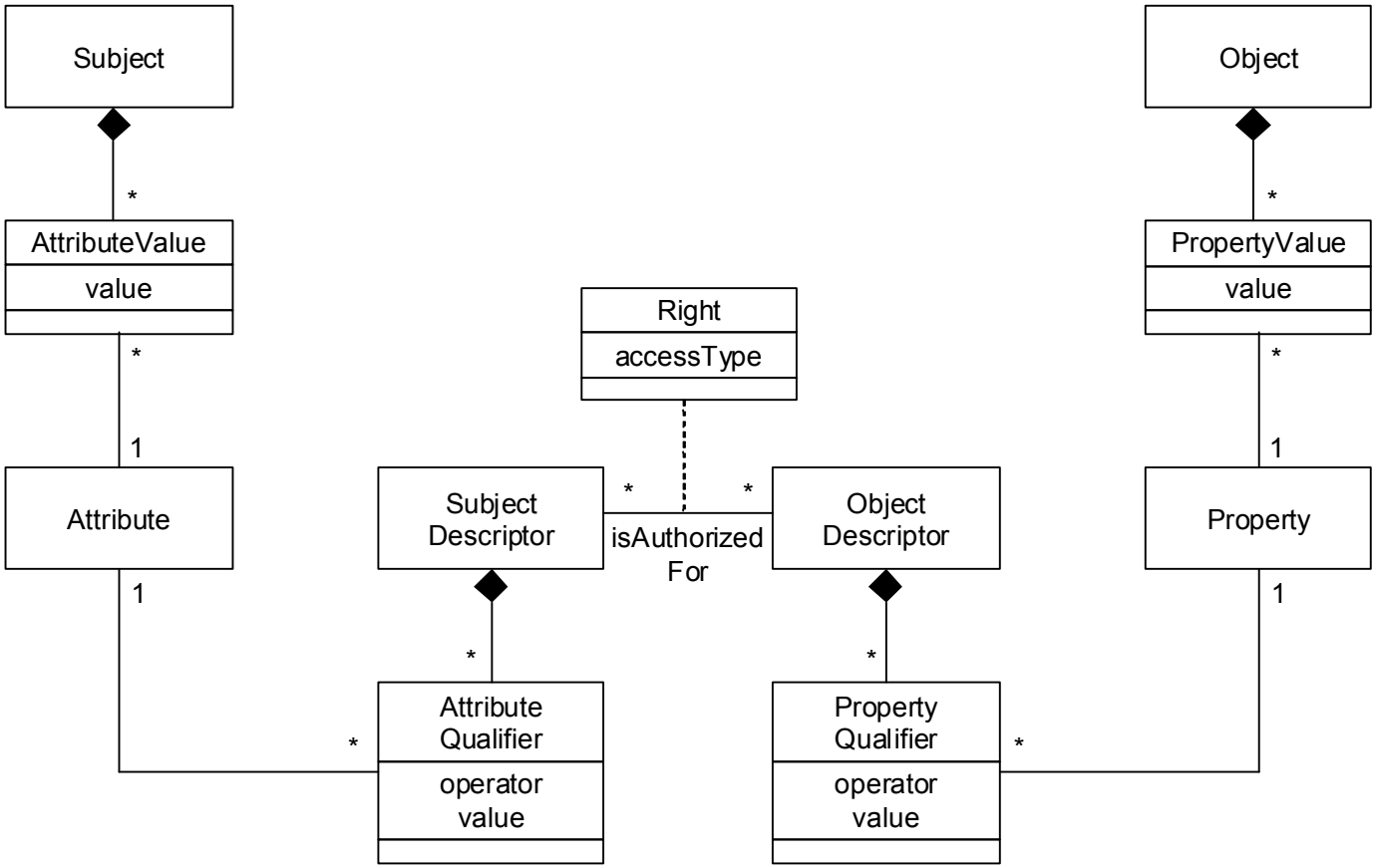
Extended RBAC pattern



Attribute-Based Access Control

- In the Internet we need to deal with non-registered users
- Determine effective subjects and objects based on attribute values

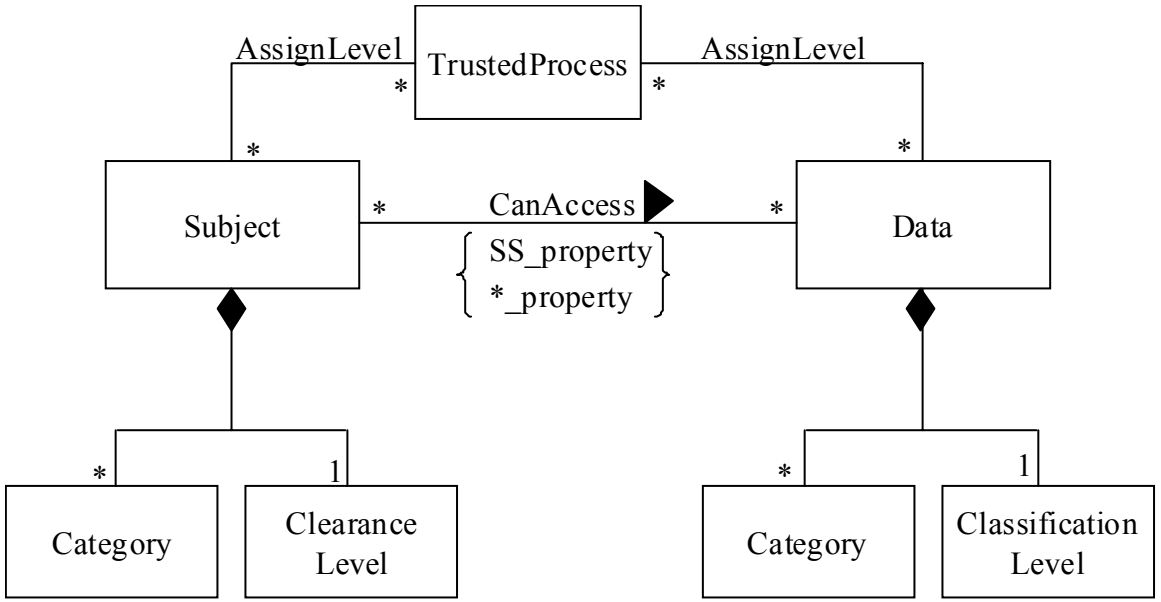
Metadata-based access control



Multilevel model

- In this model users and data are assigned classifications or clearances
- Classifications include levels (top secret, secret,…), and compartments (engDept, marketingDept,…)
- For confidentiality, access of users to data is based on rules defined by the Bell-LaPadula model, while for integrity, the rules are defined by Biba's model

Multilevel security model



Layered architecture

- The lower layers implement concrete versions of these models and enforce them
- We will look at several of these layers
- First example is from the Boundary between the network layer and the Op. system

Firewalls

- Attacks
- Types of firewalls
- Network layer firewall
- Application layer firewall
- Stateful inspection firewall

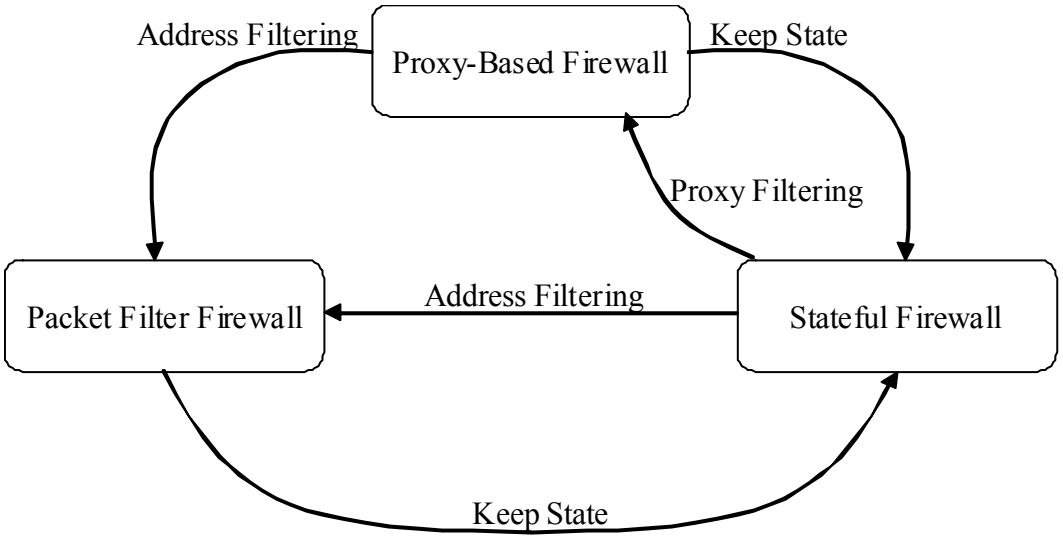
Firewall attacks

- Address spoofing
- Malformed packets
- Smuggling of attack code in packet body
- Distributed denial of service

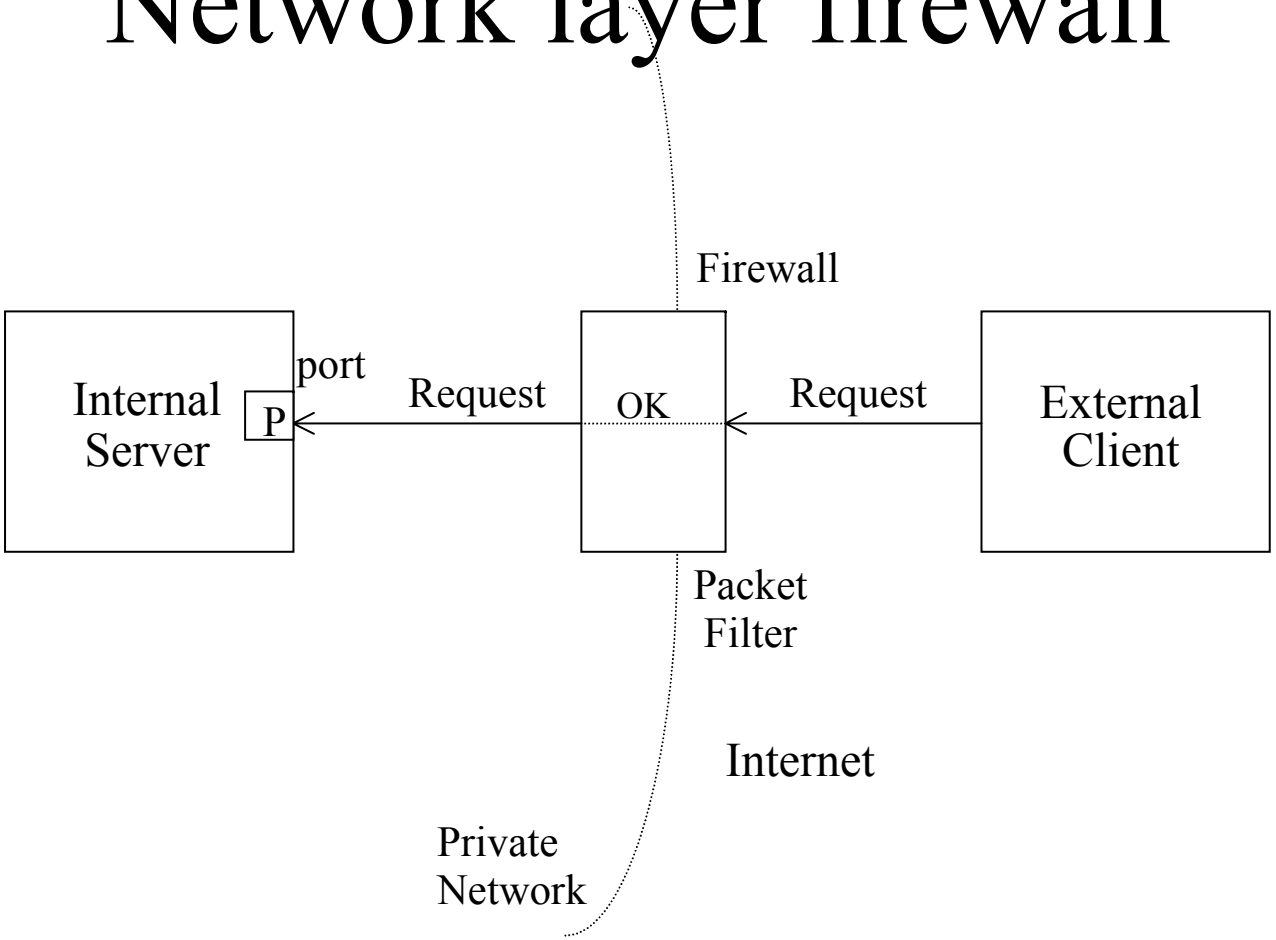
Firewalls

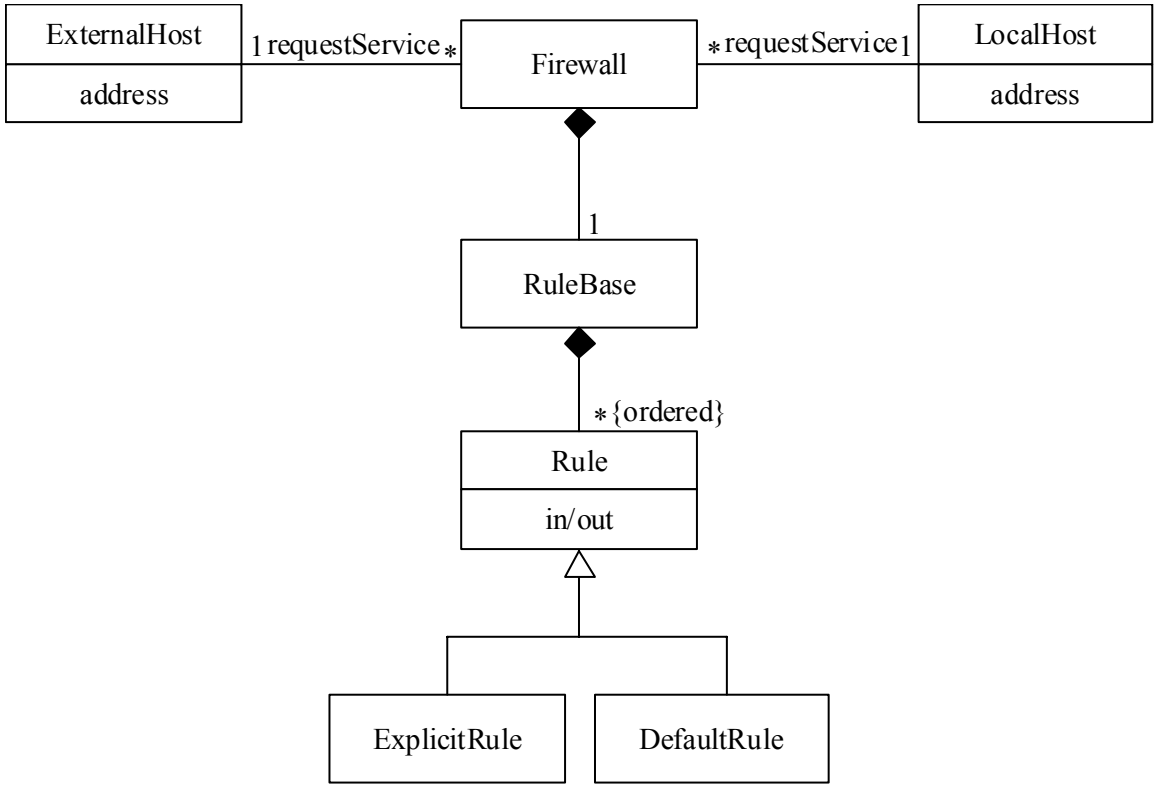
- Firewalls control access from networks to internal systems
- Network layer firewall --analyzes packets
- Application layer firewall -- uses application proxies ,supports authorization,may keep state
- Stateful inspection keeps the state of connections

Firewall patterns

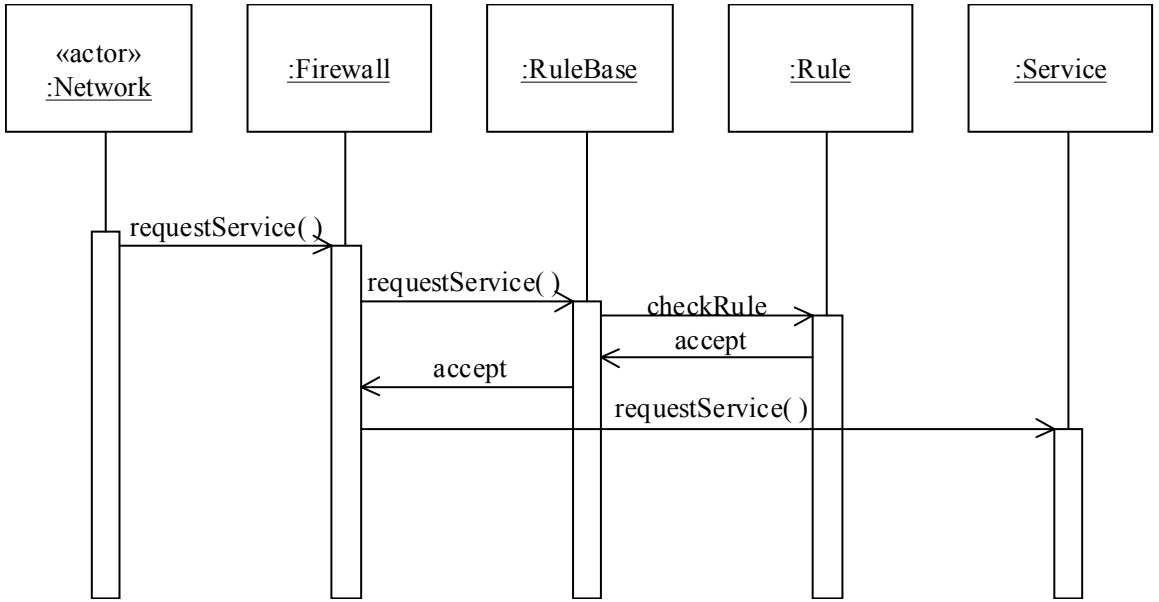


Network layer firewall




ExternalHost
address
1 requestService *
Firewall
* requestService 1
LocalHost
address
1
RuleBase
* {ordered}
Rule
in/out
ExplicitRule
DefaultRule

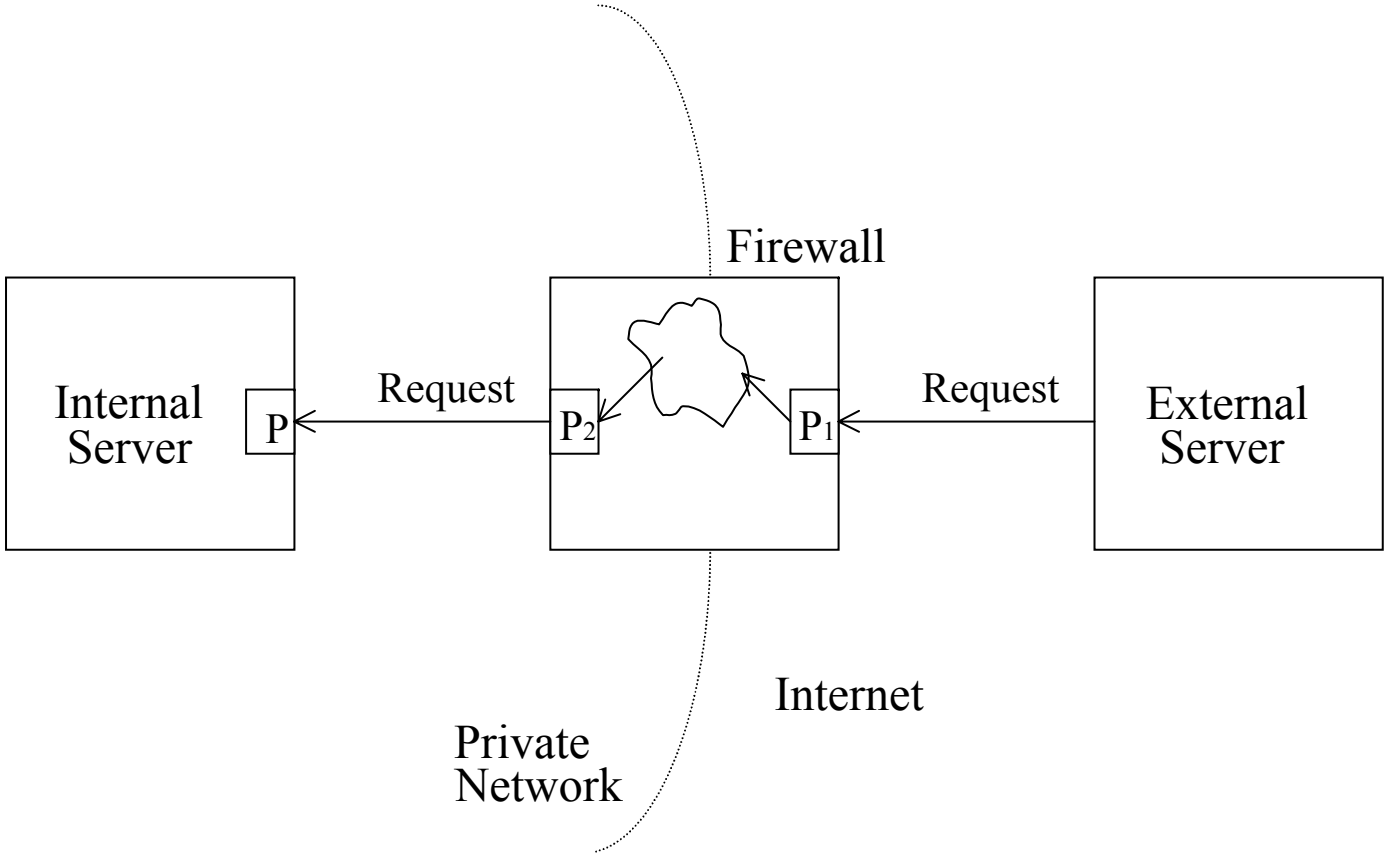
Filtering a request



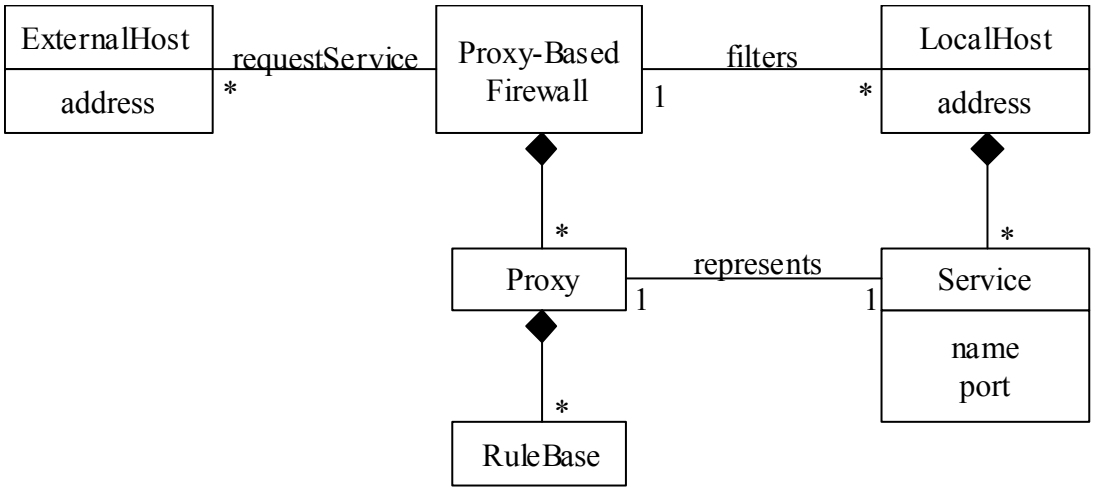
Application layer firewall

- Uses security proxies to represent services
- A variety of the Proxy pattern [GOF]
- Prevents direct access
- Analyzes application commands
- Keeps logs for later auditing
- Can keep state
- Poor scalability

Application layer firewall

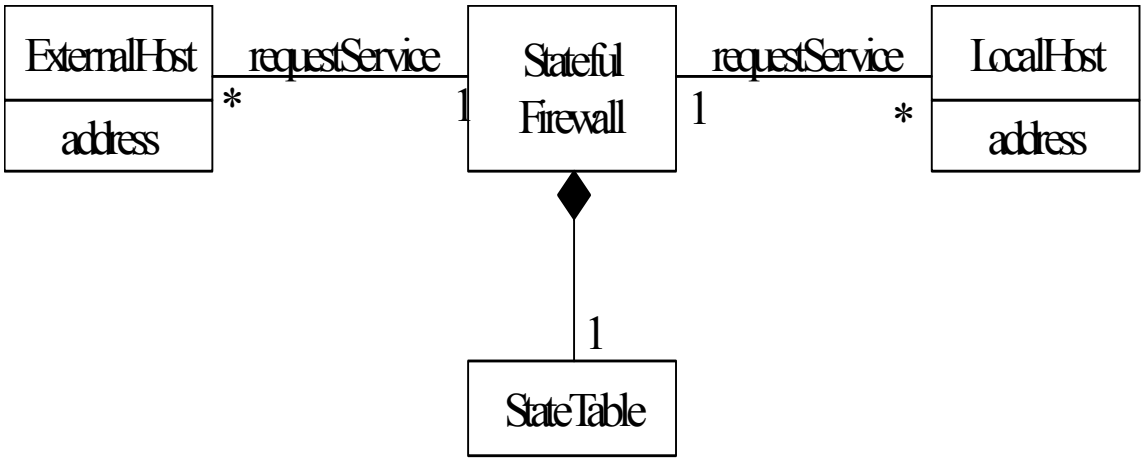


Proxy-based firewall



Stateful inspection firewall

- Based on inspection of packets
- Keeps dynamic state tables
- Can use information from the seven network layers
- Scalable
- Similar limitations about message and document contents


ExternalHost
address
requestService
*
1
Stateful
Firewall
1
requestService
1
*
LocalHost
address
1
StateTable

Operating systems

- Controls system resources
- In direct contact with hardware
- Process and processor management
- Memory management --executing programs
- Data management: persistent data
- I/O devices -- disks, communications ,…
- Controls login

OS attacks

- Remote login weaknesses
- Password guessing
- Bypass file permissions
- Scavenge memory
- Buffer overflow attacks
- Denial of service attacks (resource hogging)
- Privileged CGI scripts (in HTTP server OS)

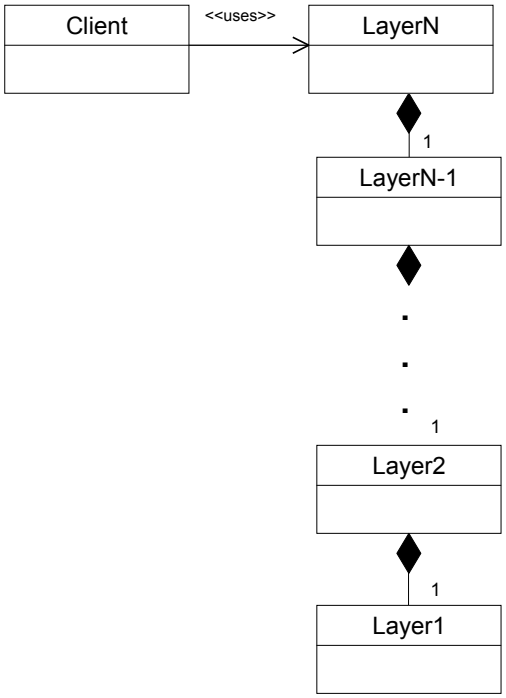
OS defenses

- Memory protection (supported by hardware)
- File protection
- Access control for I/O devices
- Requires good processor support for low overhead and to avoid bypassing of high-level mechanisms
- Capabilities and descriptors are effective mechanisms
- Firewalls to protect access to the system
- Authentication (part of login)
- A well-structured architecture

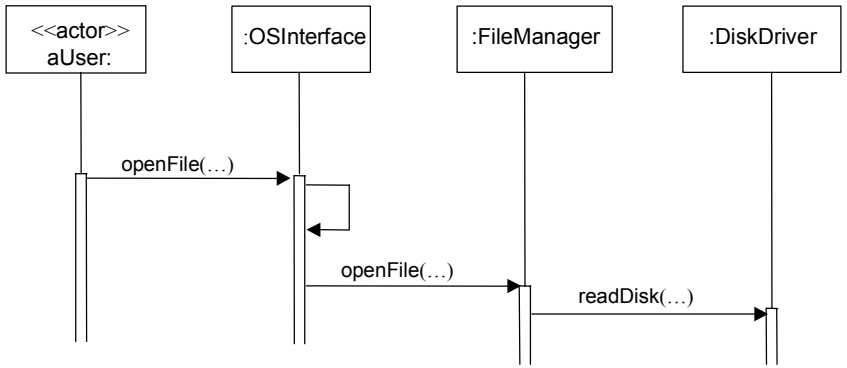
Patterns for secure architectures

- Layered OS
- Microkernel
- Virtual machine OS

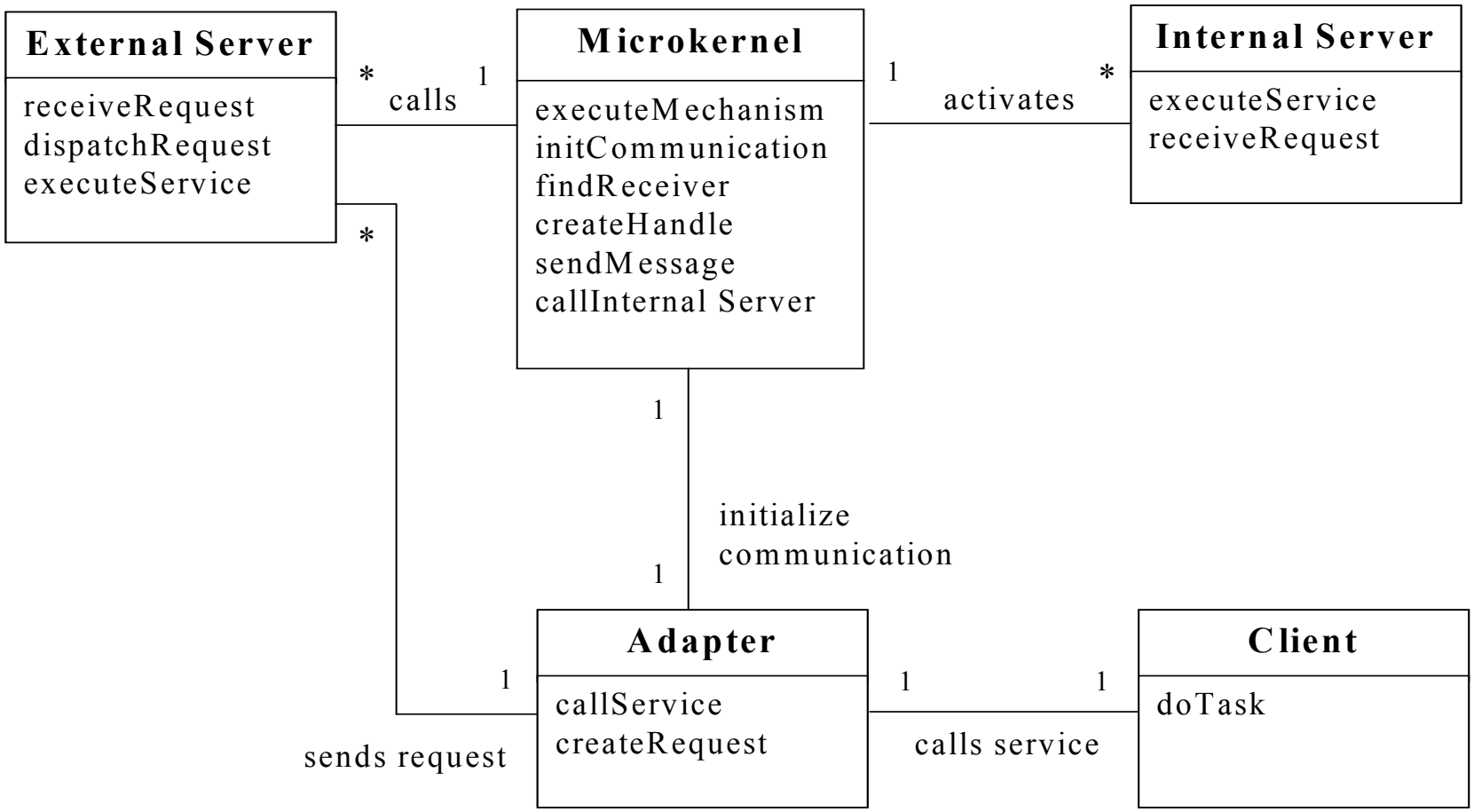
Layered OS



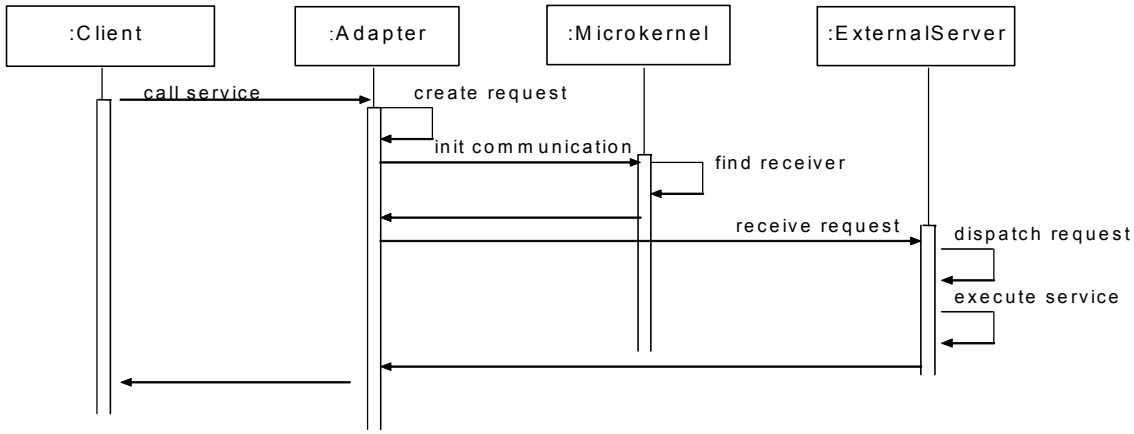
Requesting a service



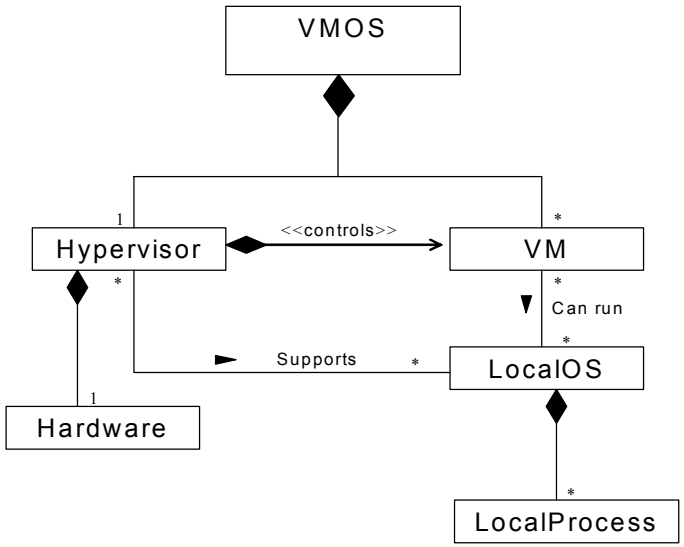
Microkernel



Requesting a service



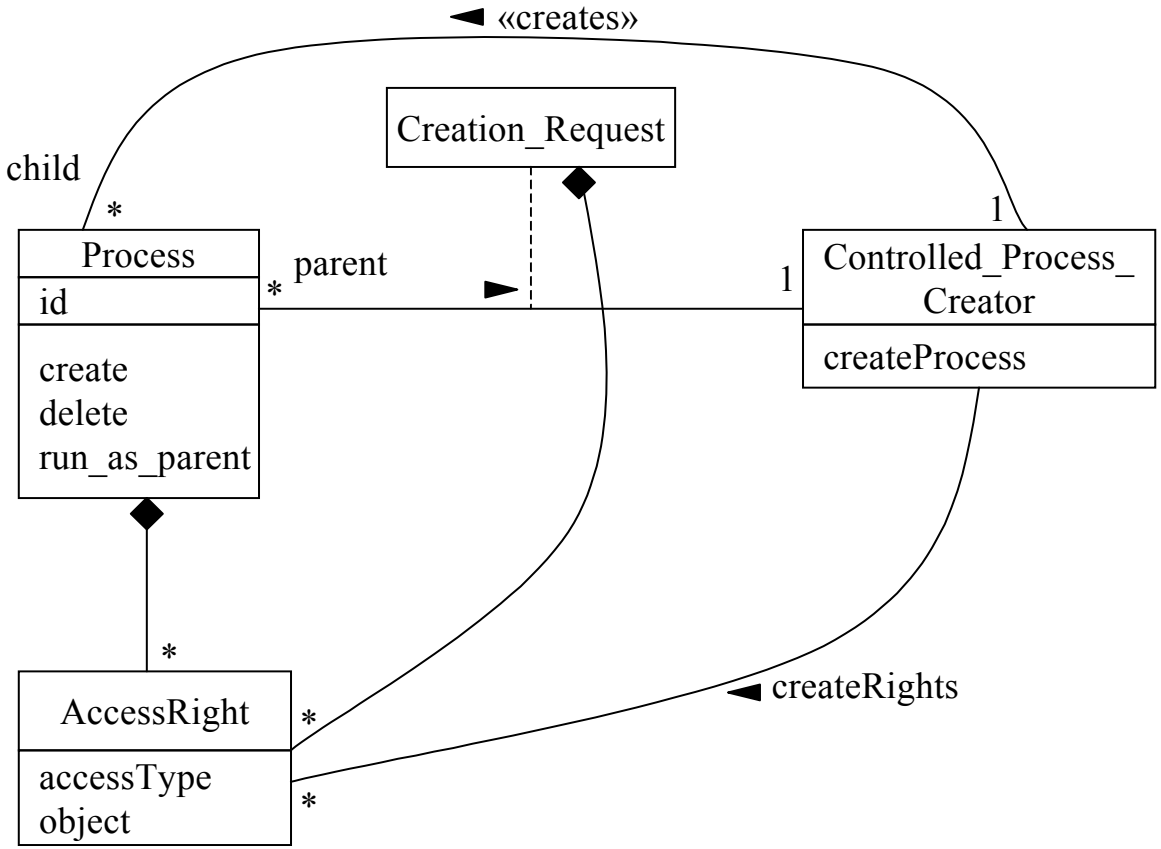
Virtual Machine OS



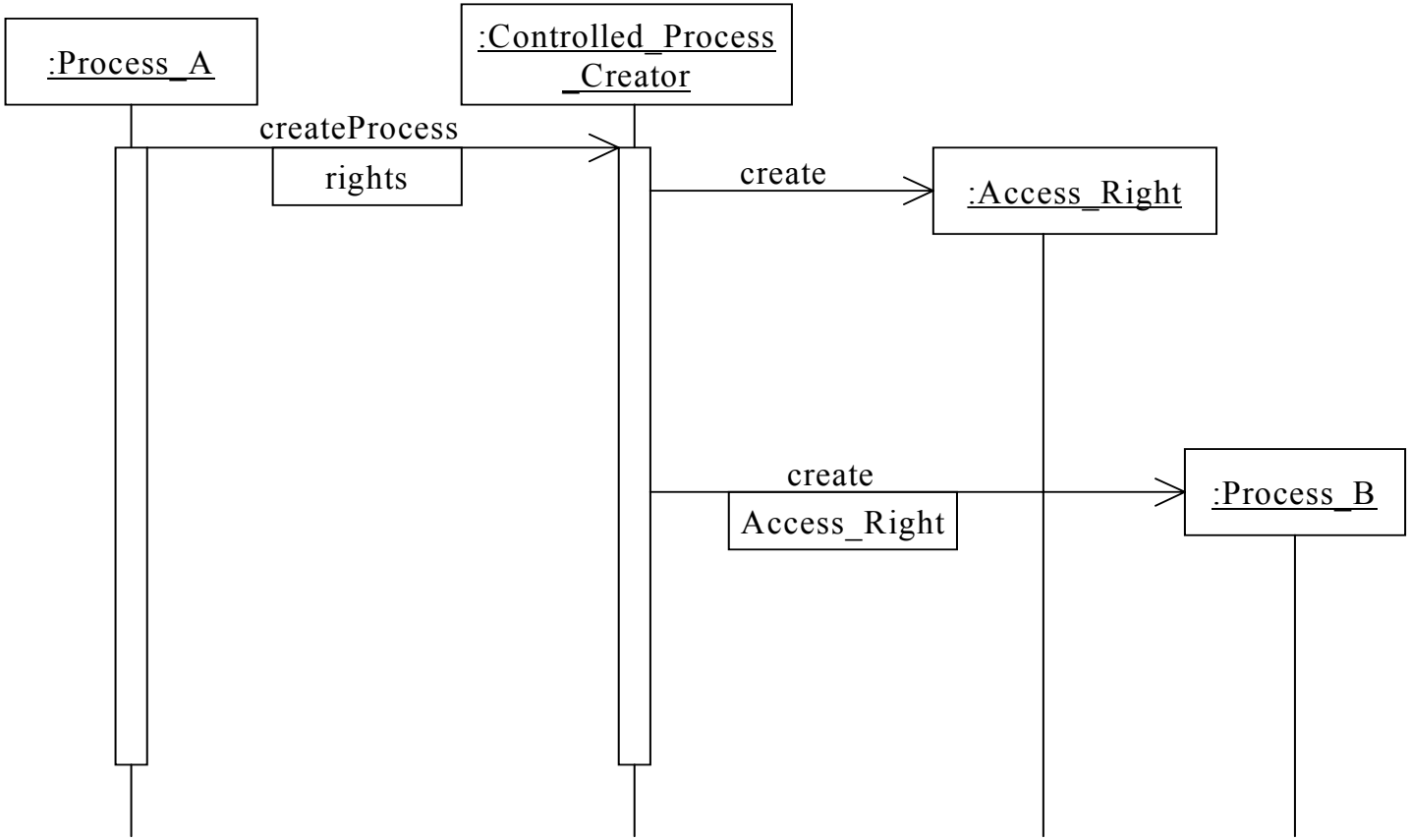
Patterns for operating systems security

- Controlled process creation
- Controlled object creation
- Authentication
- Controlled object access (reference monitor)
- File access control
- Controlled execution environment

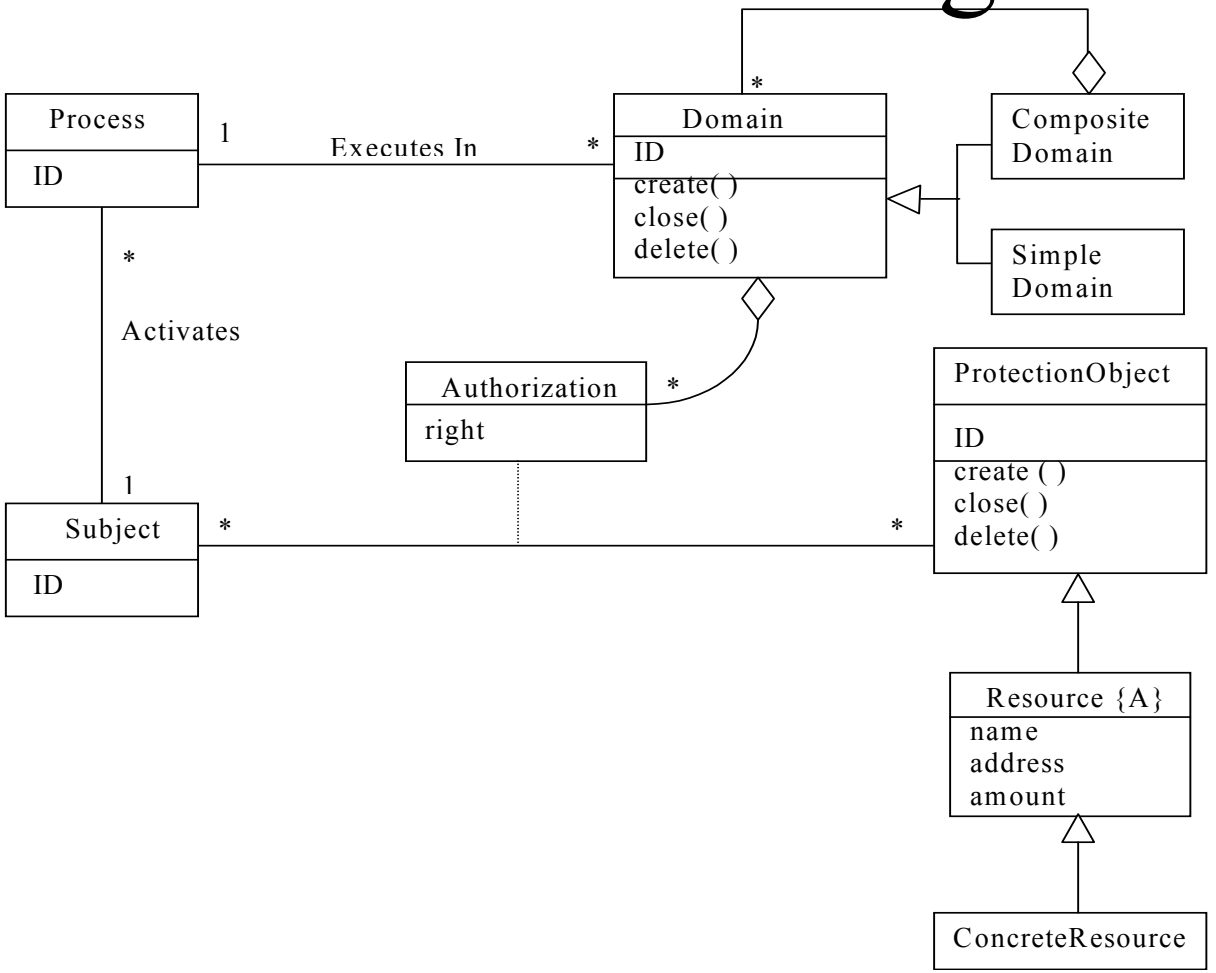
Controlled-Process Creator



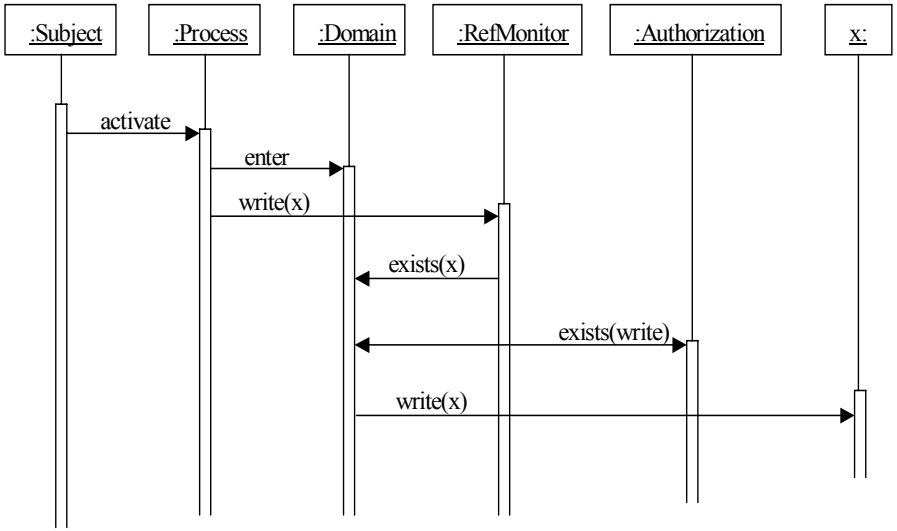
Process creation dynamics



Process/domain rights



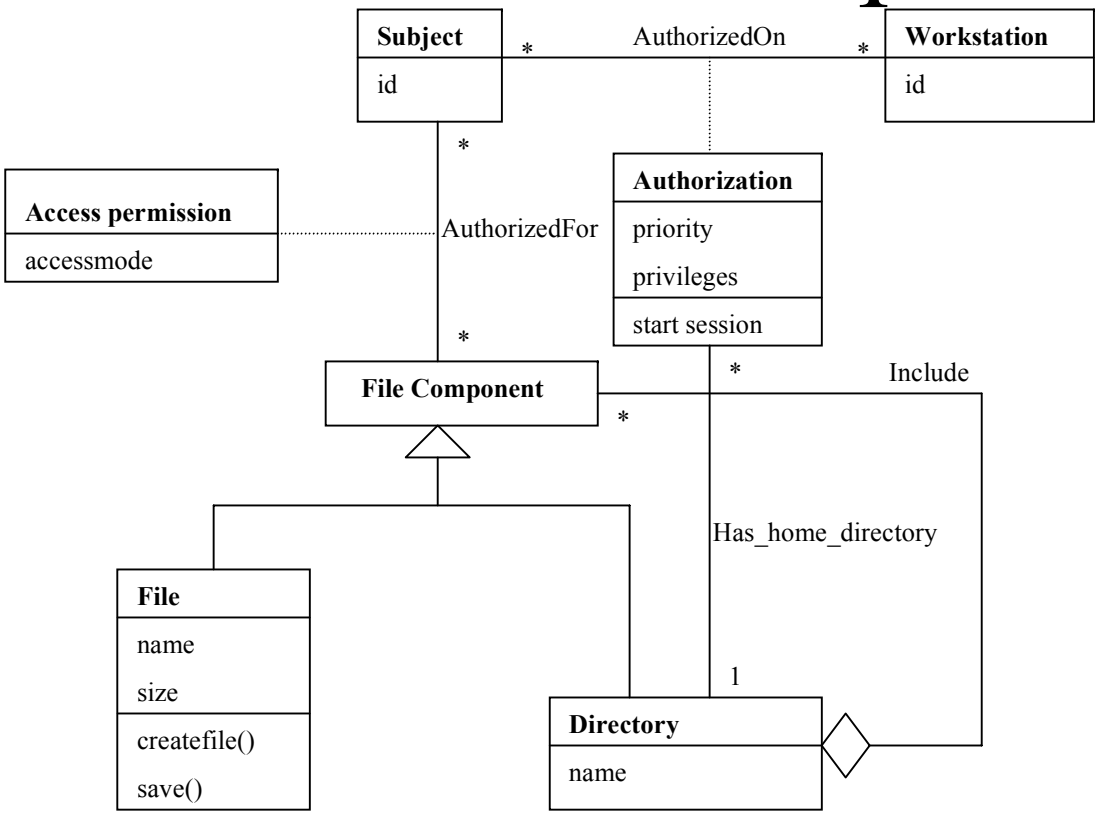
Entering a domain



Forces of file pattern

- There may be different categories of subjects, e.g., users, roles, and groups
- Subjects may be authorized to access files, directories, and workstations
- A subject has a home directory for each authorized workstation, but the same home directory can be shared among several workstation or among several subjects
- Users may be grouped for access
- Some systems may use roles instead or in addition to users as subjects
- There are many different implementations

A file authorization pattern



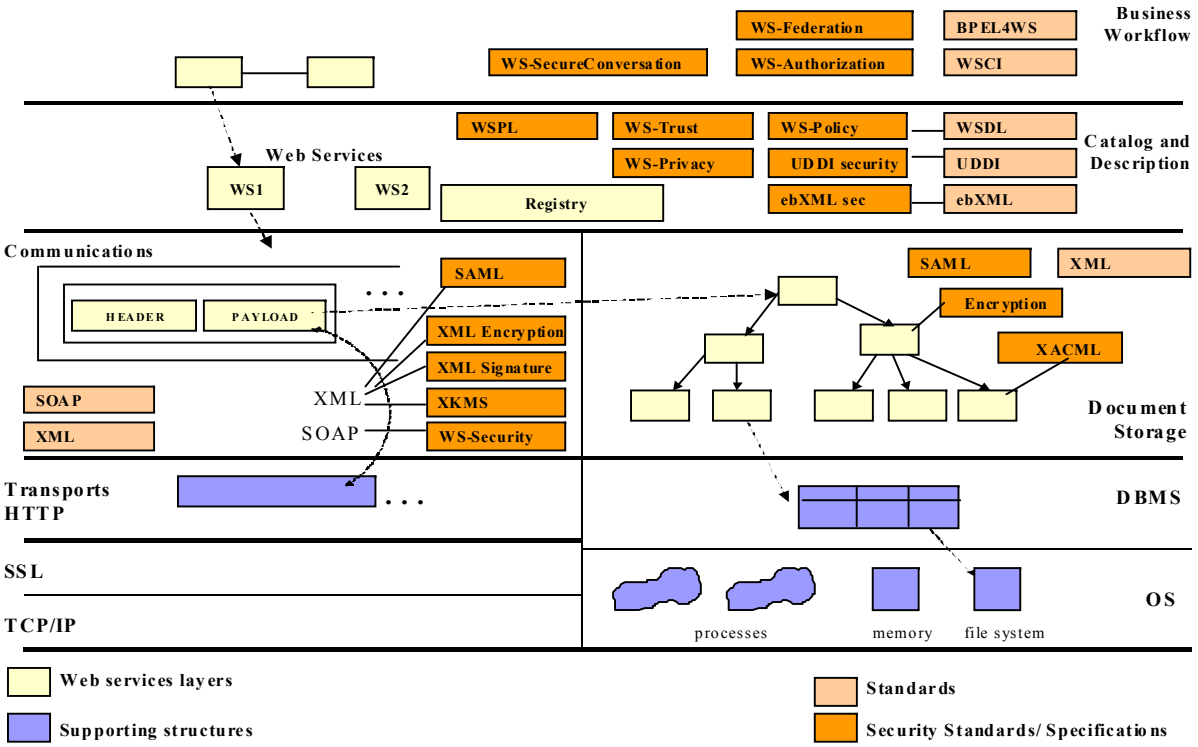
Use of subpatterns

- This pattern uses two instances of the Authorization Rule pattern
- Also uses the Composite pattern (GOF)
- A higher-level authorization rule that uses objects included in specific files can be mapped to this level for enforcement

Patterns for web services

- SAML
- XACML
- XML Firewall

Patterns can be used to compare or understand standards


Business Workflow
WS-Federation
BPEL4WS
WS-SecureConversation
WS-Authorization
WSCI
Web Services
WS1
WS2
WSPL
WS-Trust
WS-Policy
WSDL
WS-Privacy
UDDI security
UDDI
Registry
ebXML sec
ebXML
Catalog and Description
Communications
SAML
XML
HEADER
PAYLOAD
Encryption
XML Encryption
XACML
XML Signature
SOAP
XML
XKMS
SOAP
WS-Security
Document Storage
Transports HTTP
DBMS
SSL
TCP/IP
OS
processes
memory
file system
Web services layers
Supporting structures
Standards
Security Standards/ Specifications

Security Assertion Markup Language (SAML)

- Part of XML-based Security Services
- XML framework for exchanging authentication and authorization information
- SAML information can be added to XML messages

Three types of assertions

- Authentication
- Authorization
- Attributes (groups, roles,…)

Assertion Coordinator pattern

- How to apply assertions across the Internet
- Most important use is Single Sign On (SSO)
- We find classes using CRC (Class-Responsibility-Collaboration) cards

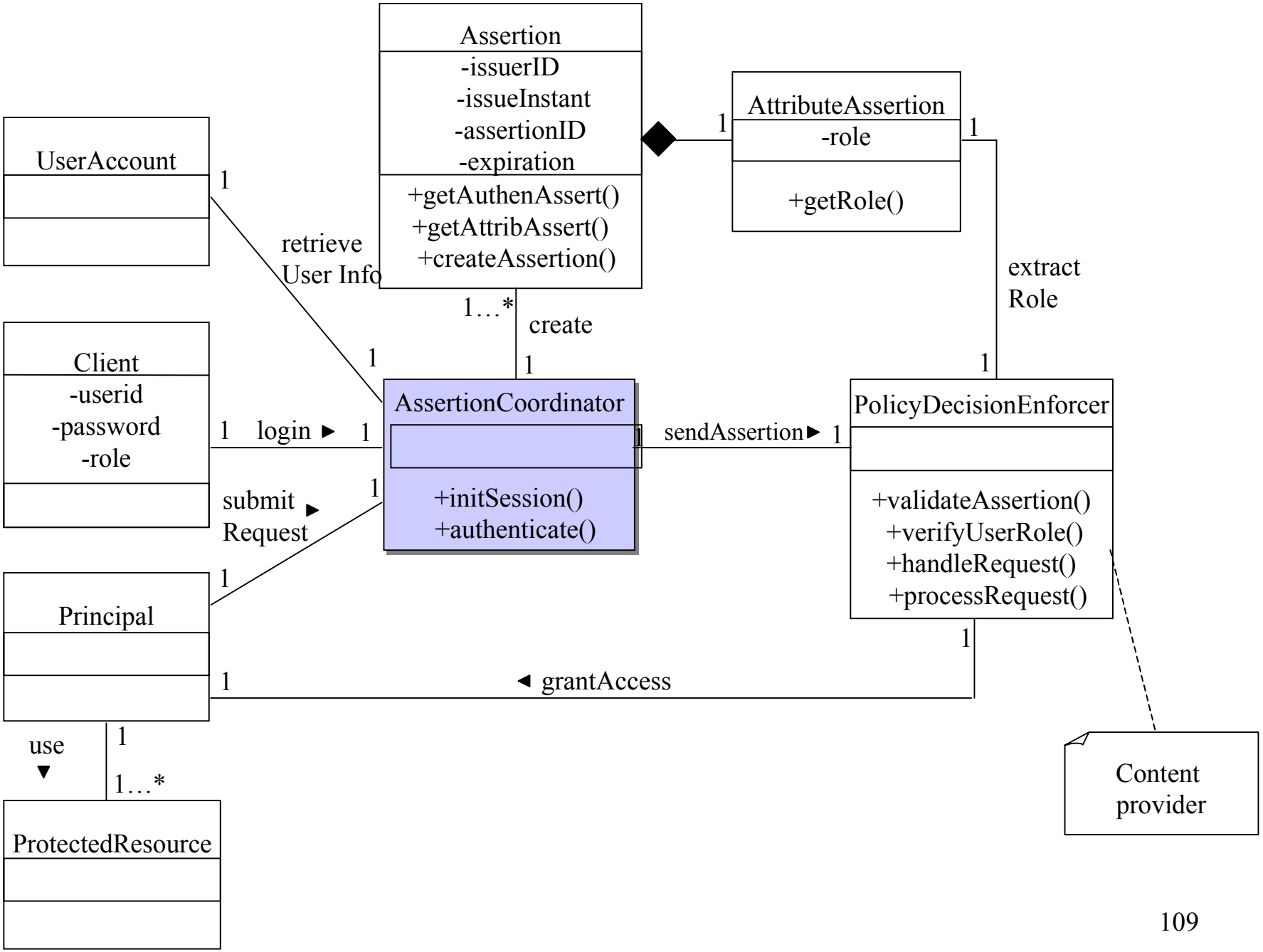
Class AssertionCoordinator	**Collaborator** •Assertion •Client •Principal •UserAccount •PolicyDecisionEnforcer
Responsibility •Authenticate users •Initiate session •Invoke creation of artifact and assertion for current user	

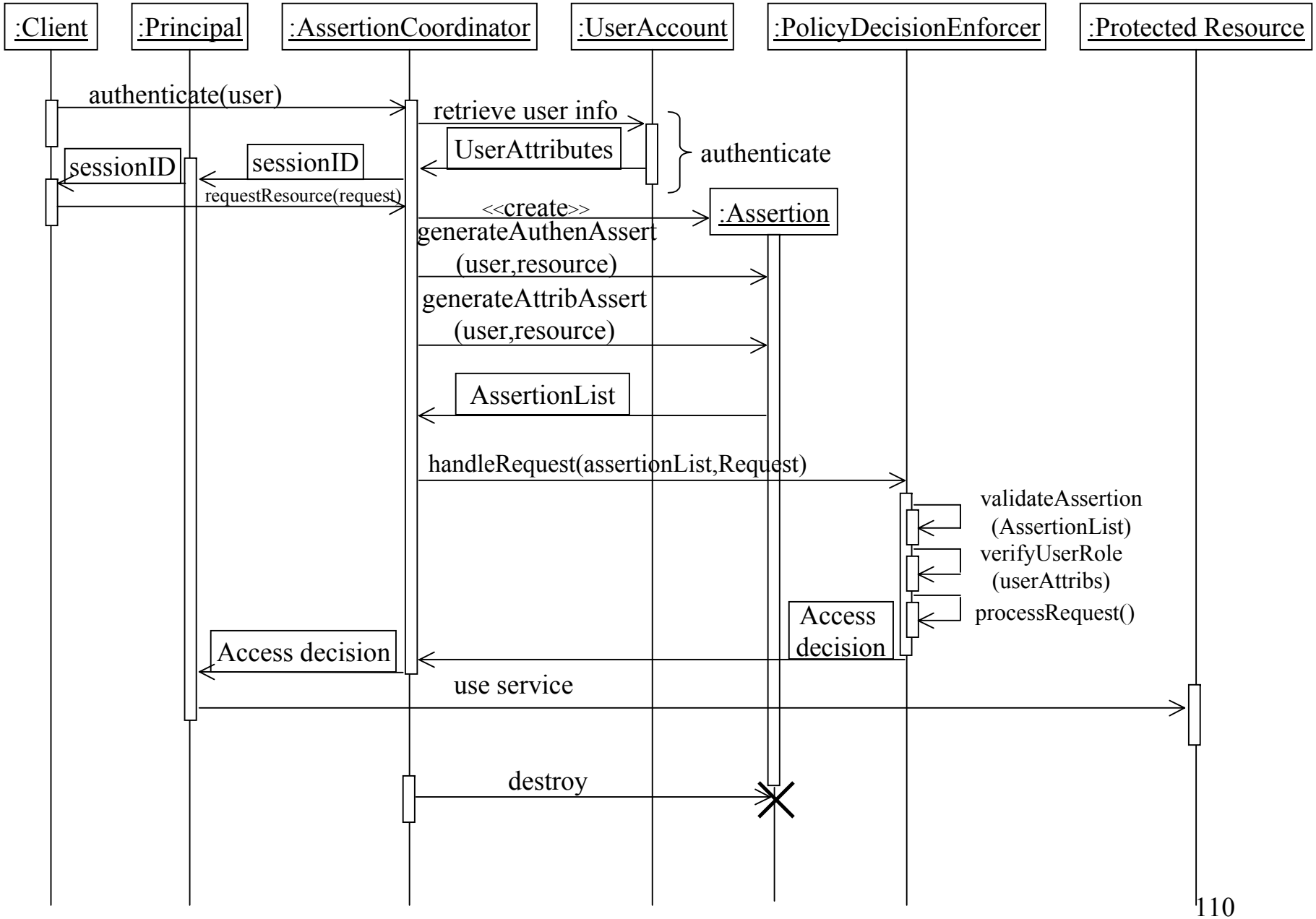
Class UserAccount	**Collaborator** •AssertionCoorcinator
Responsibility •Keep user security Information	

Class PolicyDecisionEnforcer	**Collaborator** •AssertionCoordinator •Assertion •AttributeAssertion
Responsibility •Provide content to authenticated users with the appropriate Attributes	

Class Assertion	**Collaborator** •AttributeAssertion •AssertionCoordinator
Responsibility •Represents user security data in XML form •Categorizes data in the form of authentication or attribute assertions	

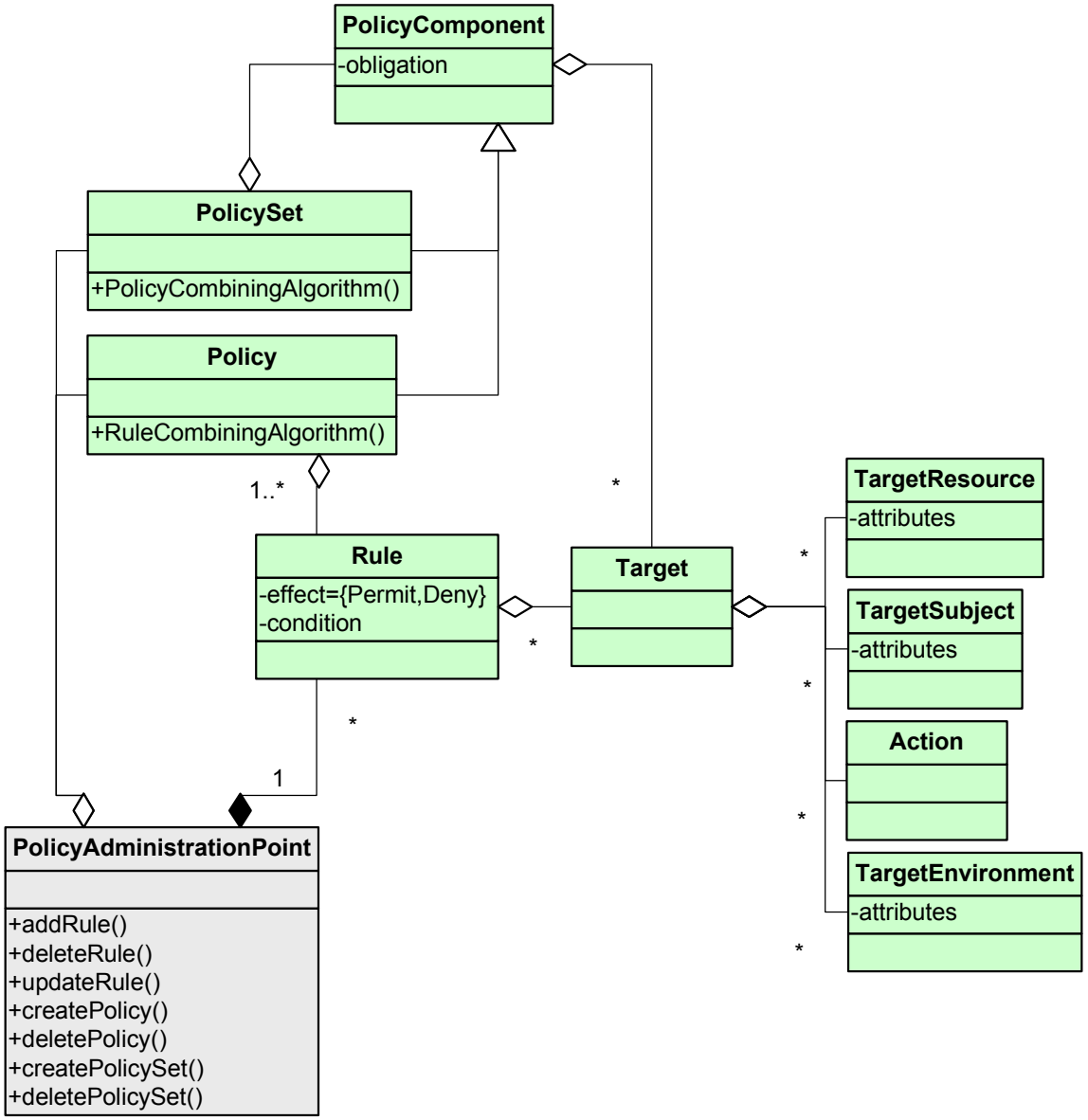
Class Principal	**Collaborator** •AssertionCoordinator •PolicyDecisionEnforcer •ProtectedResource
Responsibility •Identify clients authenticated by the AssertionCoordinator •Request protected resources •Fill order requests	

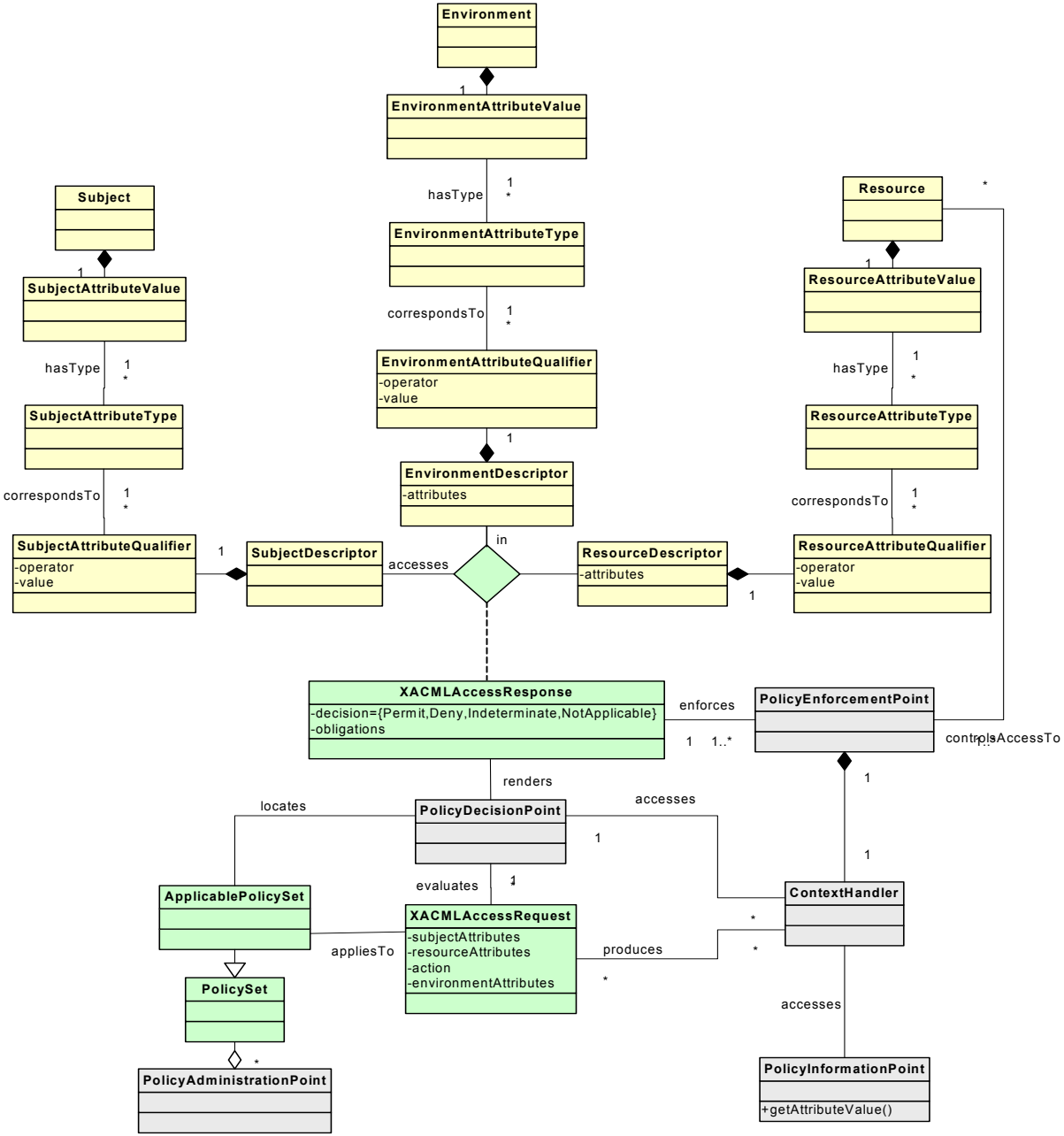

Assertion
-issuerID
-issueInstant
-assertionID
-expiration
+getAuthenAssert()
+getAttribAssert()
+createAssertion()
AttributeAssertion
-role
+getRole()
UserAccount
retrieve
User Info
extract
Role
1…*
create
Client
-userid
-password
-role
AssertionCoordinator
+initSession()
+authenticate()
PolicyDecisionEnforcer
+validateAssertion()
+verifyUserRole()
+handleRequest()
+processRequest()
login
sendAssertion
submit
Request
Principal
grantAccess
use
1…*
ProtectedResource
Content
provider


:Client
:Principal
:AssertionCoordinator
:UserAccount
:PolicyDecisionEnforcer
:Protected Resource
authenticate(user)
retrieve user info
UserAttributes
authenticate
sessionID
sessionID
requestResource(request)
<<create>>
:Assertion
generateAuthenAssert
(user,resource)
generateAttribAssert
(user,resource)
AssertionList
handleRequest(assertionList,Request)
validateAssertion
(AssertionList)
verifyUserRole
(userAttribs)
processRequest()
Access
decision
Access decision
use service
destroy

XACML

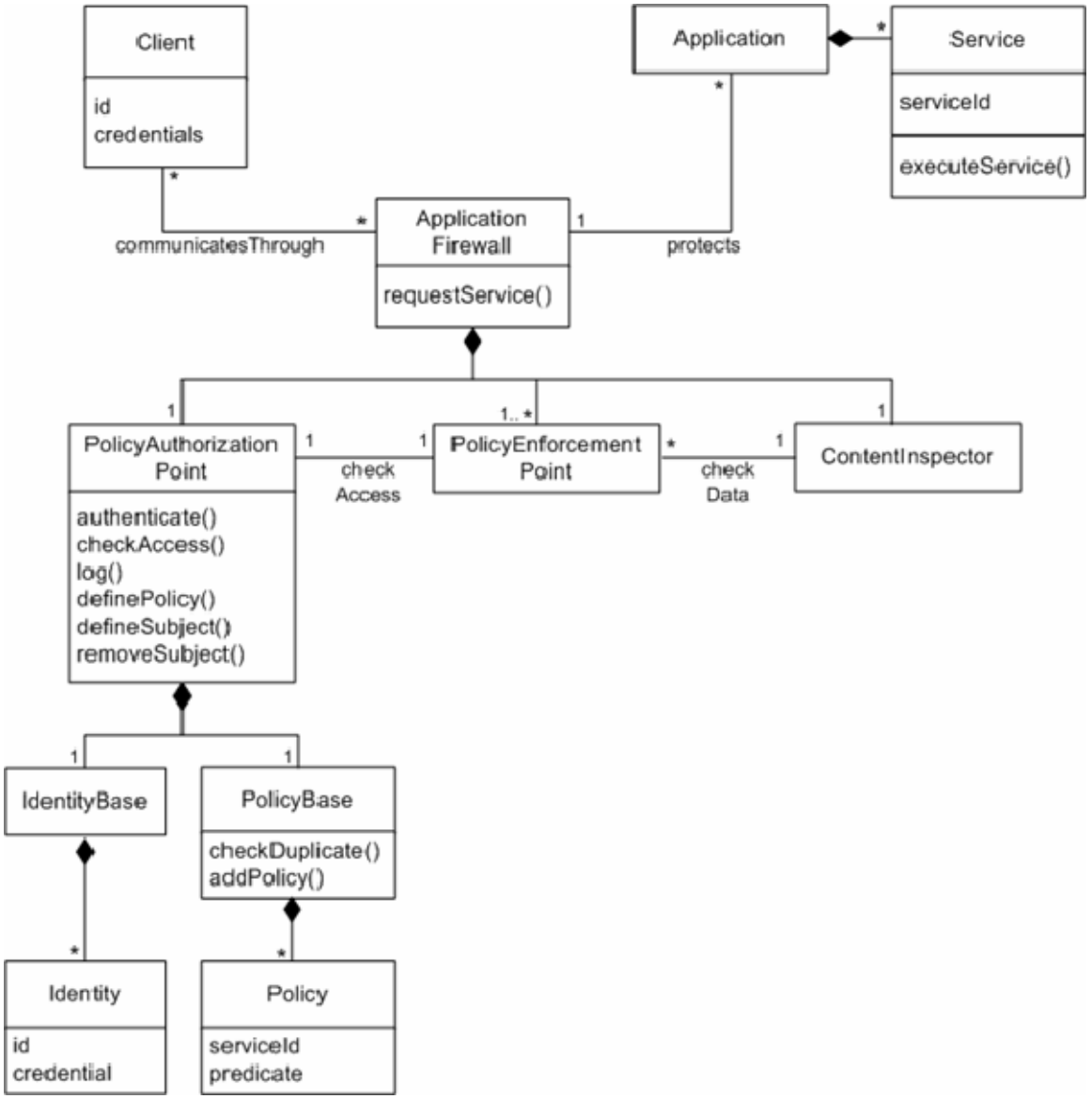
- Special technical committee of OASIS
- Specification of policies for information access over the Internet and their enforcement
- Combines work of IBM Tokyo and University of Milano, Italy.
- Implemented by Sun in early 2003

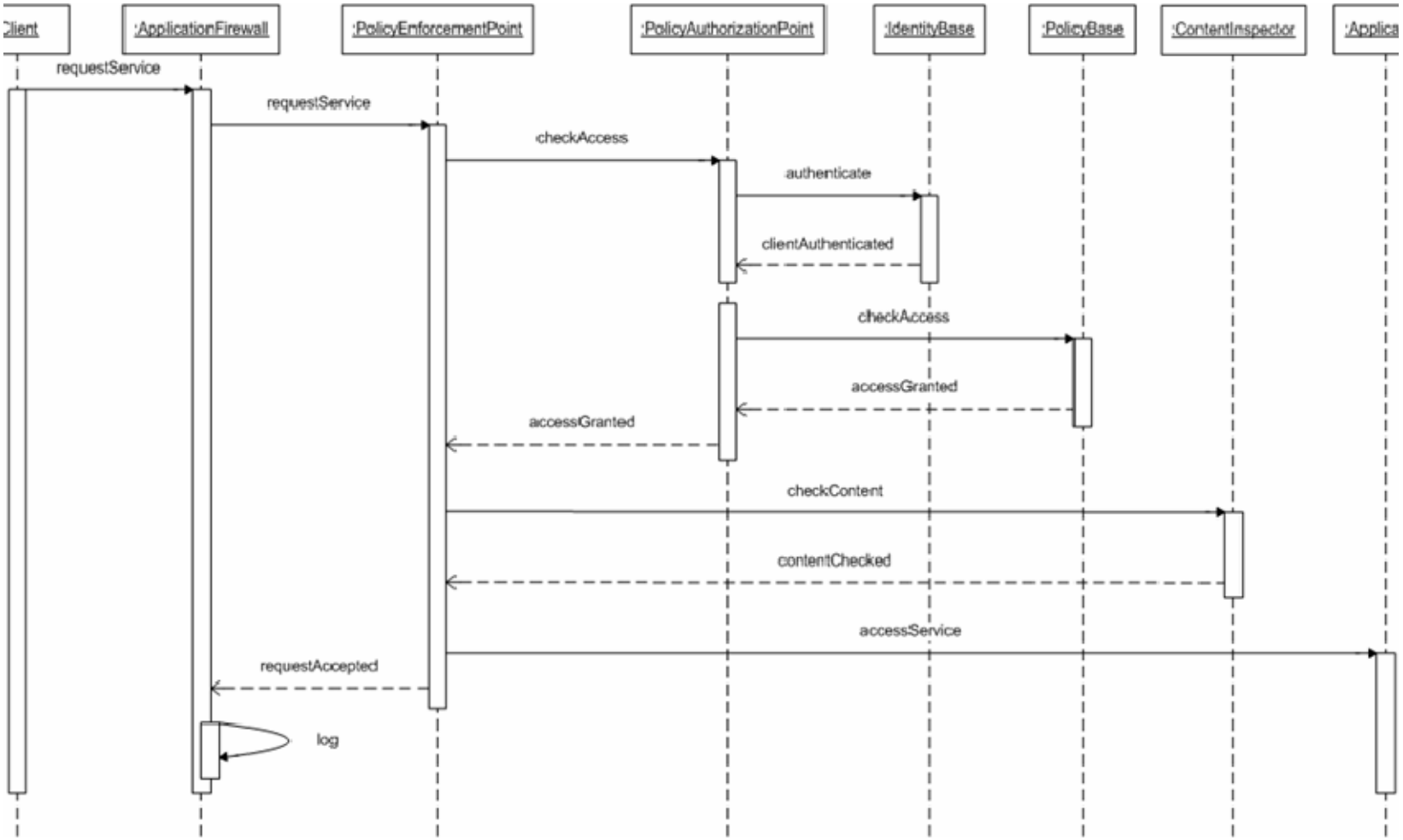

PolicyComponent
-obligation
PolicySet
+PolicyCombiningAlgorithm()
Policy
+RuleCombiningAlgorithm()
1..*
*
Rule
-effect={Permit,Deny}
-condition
Target
*
*
*
1
*
*
*
*
TargetResource
-attributes
TargetSubject
-attributes
Action
TargetEnvironment
-attributes
PolicyAdministrationPoint
+addRule()
+deleteRule()
+updateRule()
+createPolicy()
+deletePolicy()
+createPolicySet()
+deletePolicySet()


Environment
EnvironmentAttributeValue
hasType
EnvironmentAttributeType
correspondsTo
EnvironmentAttributeQualifier
-operator
-value
EnvironmentDescriptor
-attributes
Subject
SubjectAttributeValue
hasType
SubjectAttributeType
correspondsTo
SubjectAttributeQualifier
-operator
-value
SubjectDescriptor
accesses
in
ResourceDescriptor
-attributes
Resource
ResourceAttributeValue
hasType
ResourceAttributeType
correspondsTo
ResourceAttributeQualifier
-operator
-value
XACMLAccessResponse
-decision={Permit,Deny,Indeterminate,NotApplicable}
-obligations
enforces
PolicyEnforcementPoint
controlsAccessTo
renders
locates
PolicyDecisionPoint
accesses
evaluates
ApplicablePolicySet
XACMLAccessRequest
-subjectAttributes
-resourceAttributes
-action
-environmentAttributes
appliesTo
produces
ContextHandler
PolicySet
PolicyAdministrationPoint
accesses
PolicyInformationPoint
+getAttributeValue()

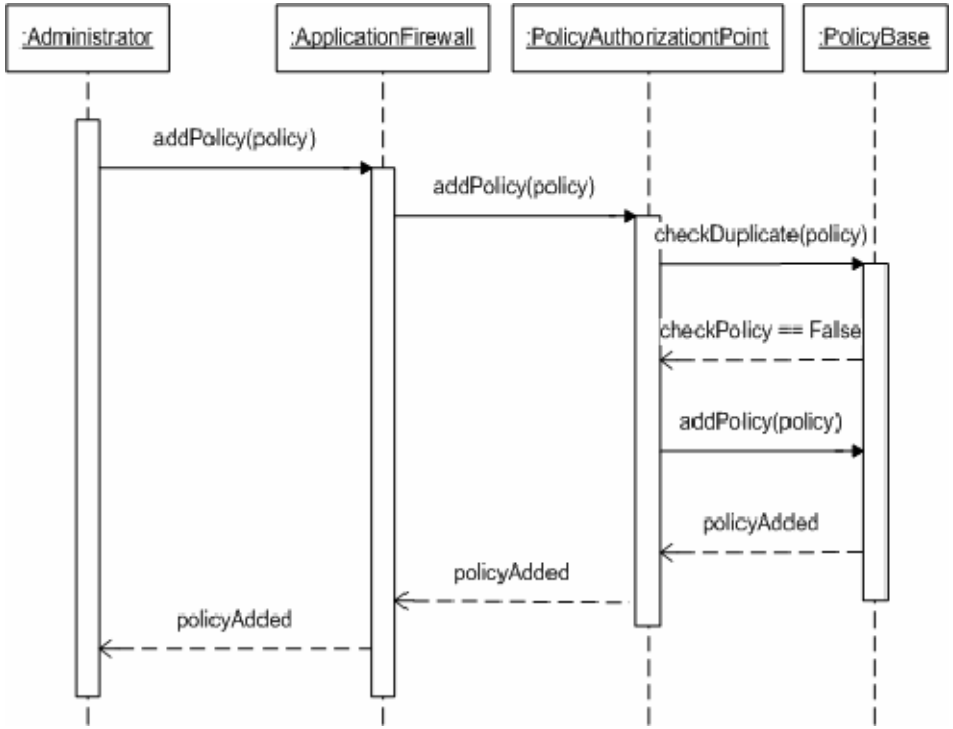
Application firewall

- XML firewall is a special case of it
- Controls input/output from distributed applications
- Can filter wrong commands, wrong type or length parameters, wrong sequences


Client
id
credentials
Application
Service
serviceId
executeService()
communicatesThrough
Application Firewall
requestService()
protects
PolicyAuthorization Point
authenticate()
checkAccess()
log()
definePolicy()
defineSubject()
removeSubject()
check Access
PolicyEnforcement Point
check Data
ContentInspector
IdentityBase
PolicyBase
checkDuplicate()
addPolicy()
Identity
id
credential
Policy
serviceId
predicate

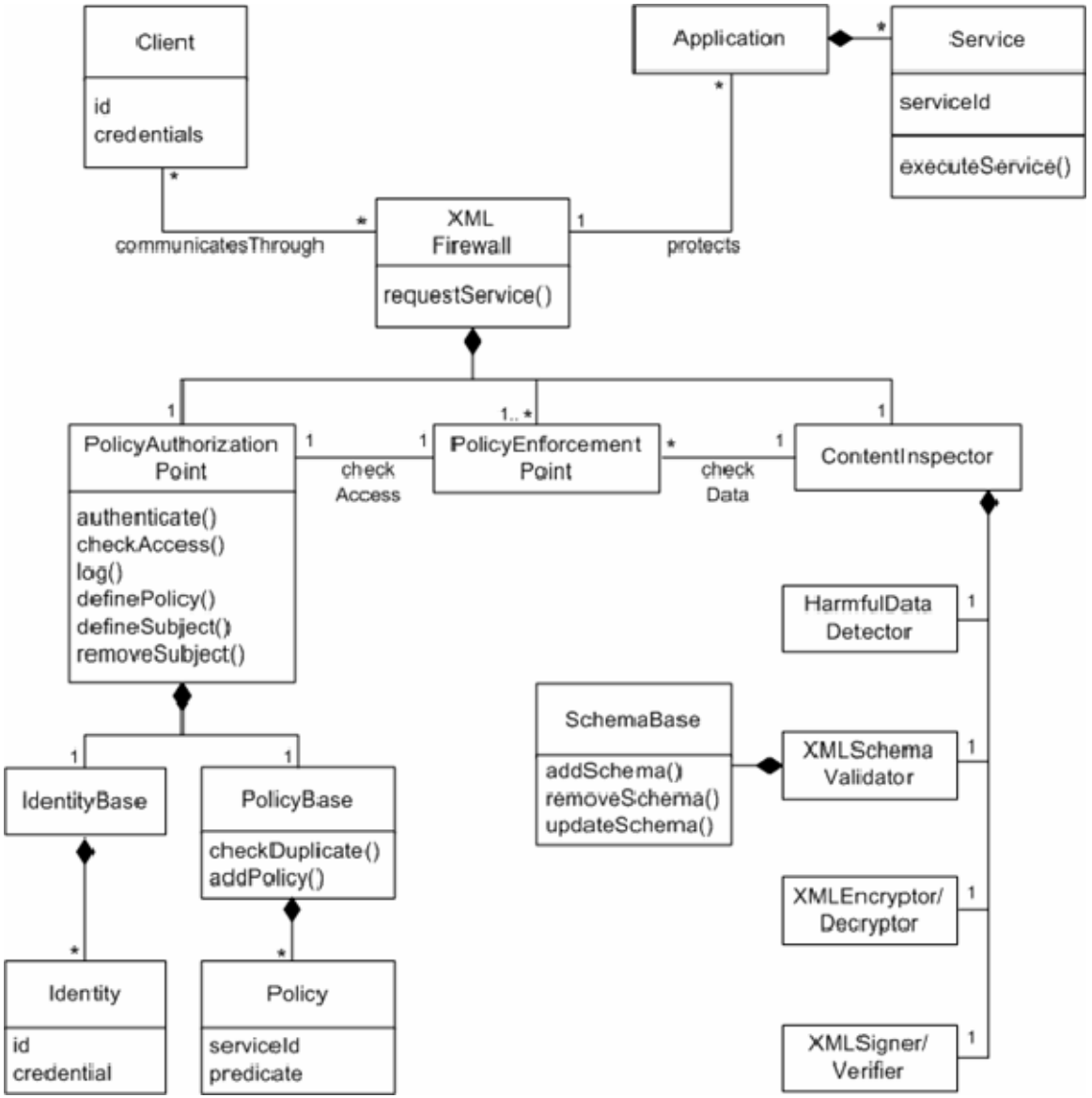

:Client
:ApplicationFirewall
:PolicyEnforcementPoint
:PolicyAuthorizationPoint
:IdentityBase
:PolicyBase
:ContentInspector
:Applica
requestService
requestService
checkAccess
authenticate
clientAuthenticated
checkAccess
accessGranted
accessGranted
checkContent
contentChecked
accessService
requestAccepted
log

Adding a new rule



XML firewall

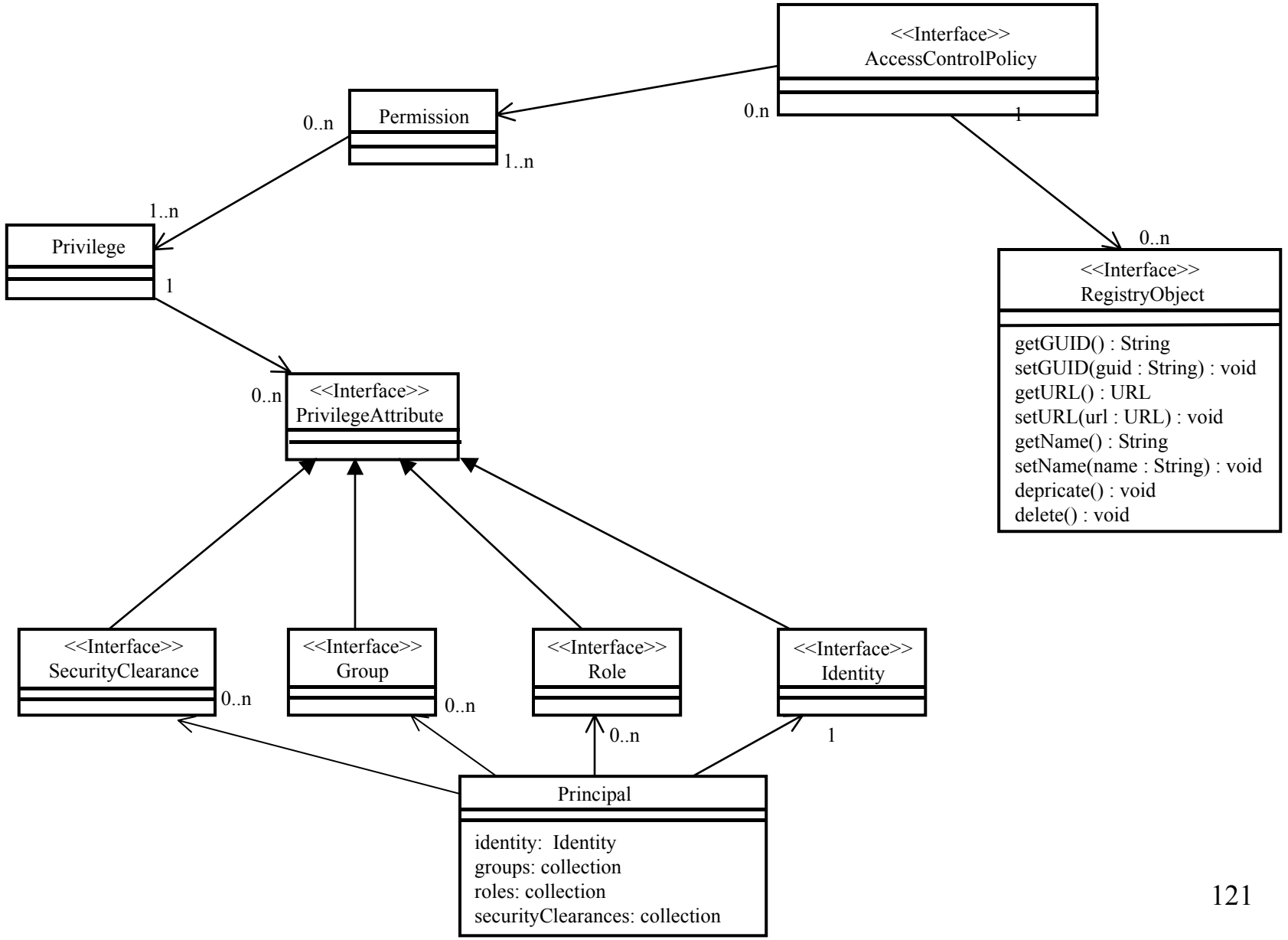
- Controls input/output of XML applications
- Well-formed documents (schema as reference)
- Harmful data (wrong type or length)
- Encryption/decryption
- Signed documents


Client
id
credentials
Application
Service
serviceId
executeService()
communicatesThrough
XML Firewall
requestService()
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

Security in ebXML

- Proposal for registry security (May 2001)
- Requirements for authentication, integrity, and confidentiality
- Each request must be authenticated
- Policy: any known entity can publish and anyone can view
- UML model for registry security

ebXML Registry Security model

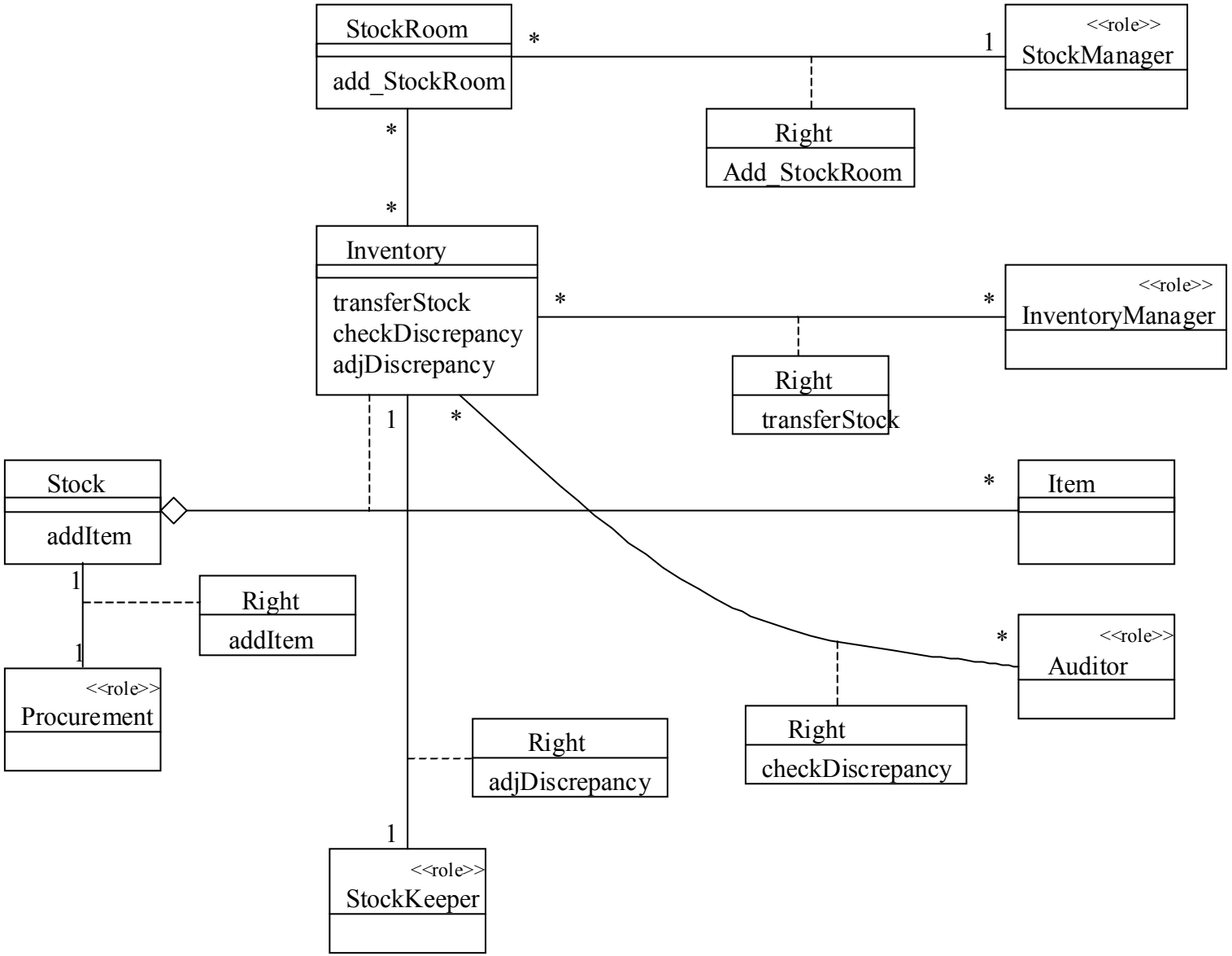


Application security

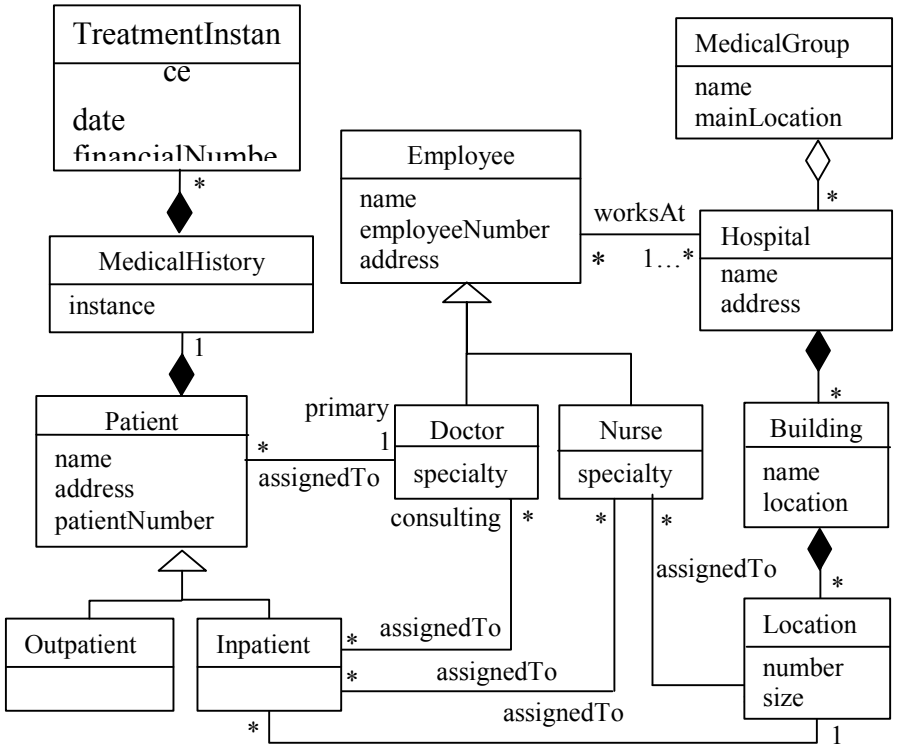
- Secure analysis patterns
- Stock manager
- Patient records
- Medical information

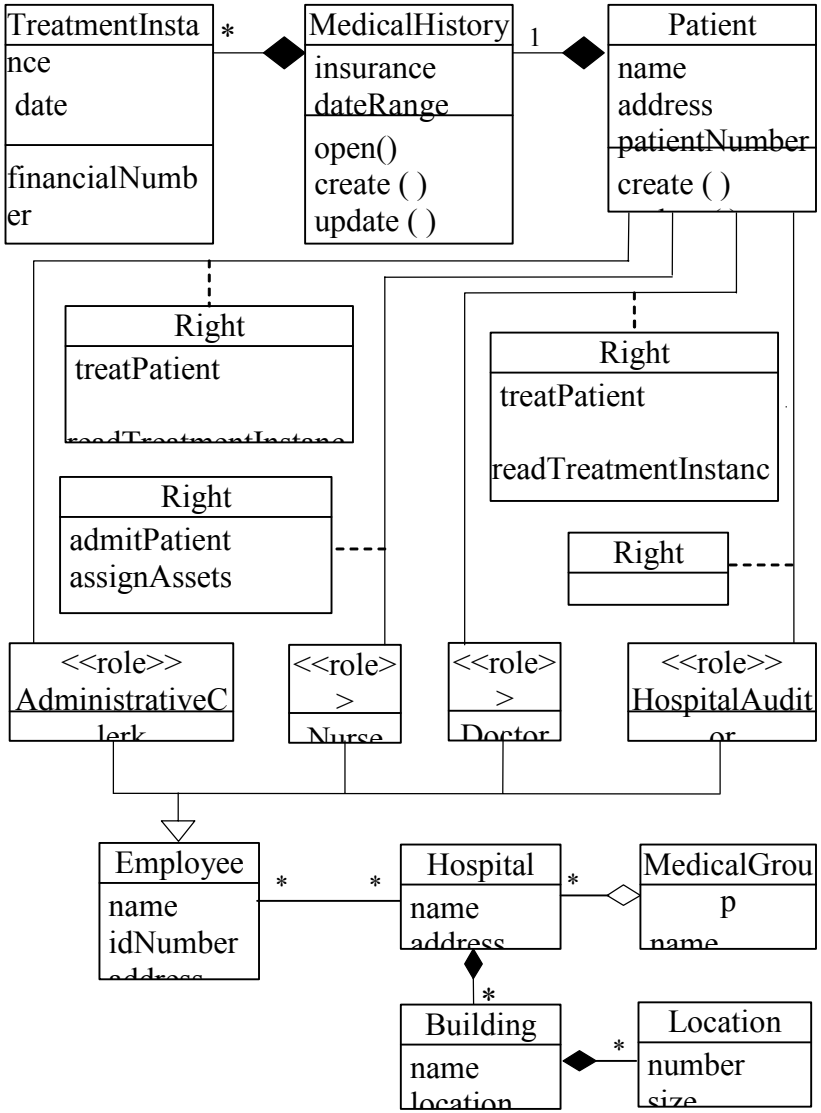
Secure analysis patterns

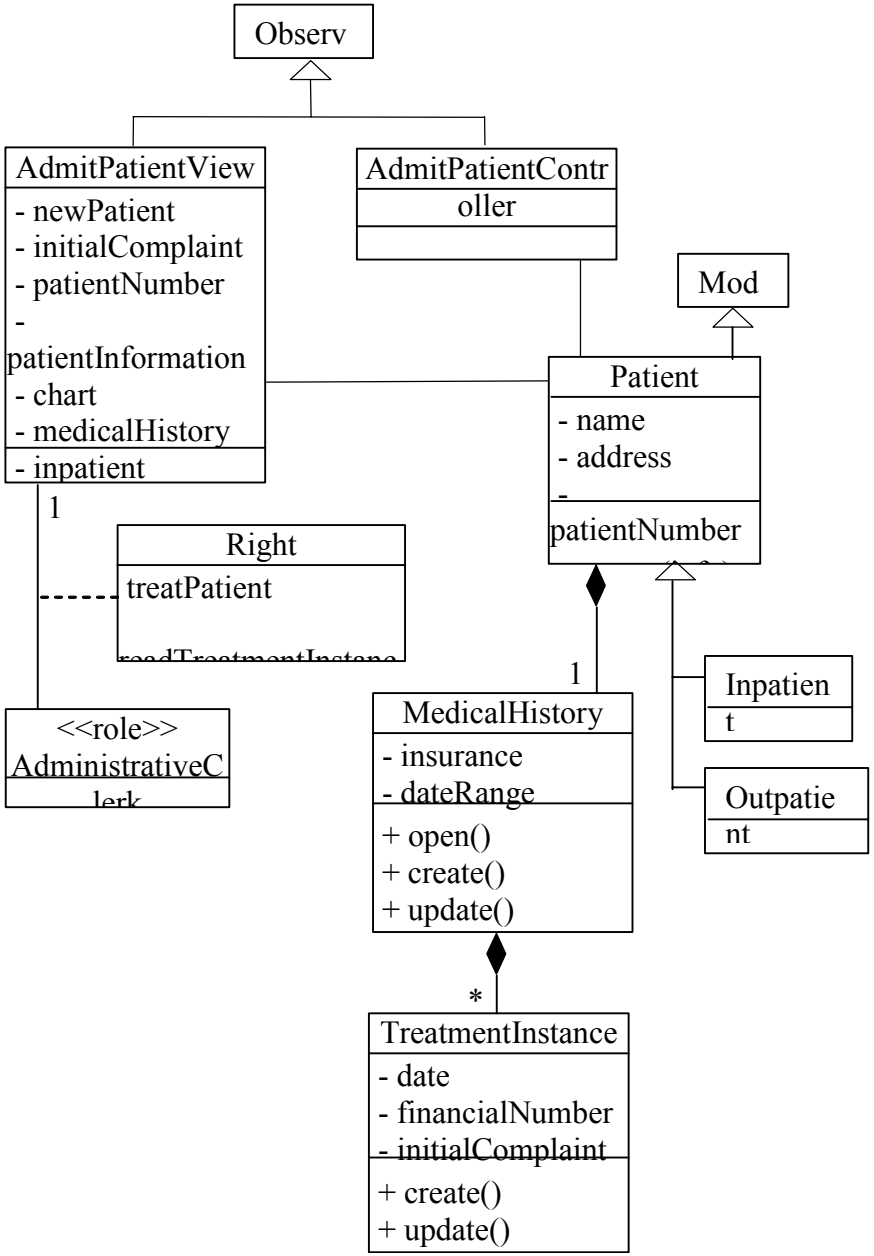
- A Semantic Analysis Pattern (SAP) describes a semantic unit
- We can combine a SAP with security patterns to get a secure semantic unit
- Can be used to build secure applications
- Example: authorized inventory system


StockRoom
add_StockRoom
*
1
<<role>>
StockManager
Right
Add_StockRoom
*
*
Inventory
transferStock
checkDiscrepancy
adjDiscrepancy
*
*
<<role>>
InventoryManager
Right
transferStock
1
*
Stock
addItem
*
Item
1
Right
addItem
1
<<role>>
Procurement
*
<<role>>
Auditor
Right
adjDiscrepancy
Right
checkDiscrepancy
1
<<role>>
StockKeeper

Patient records

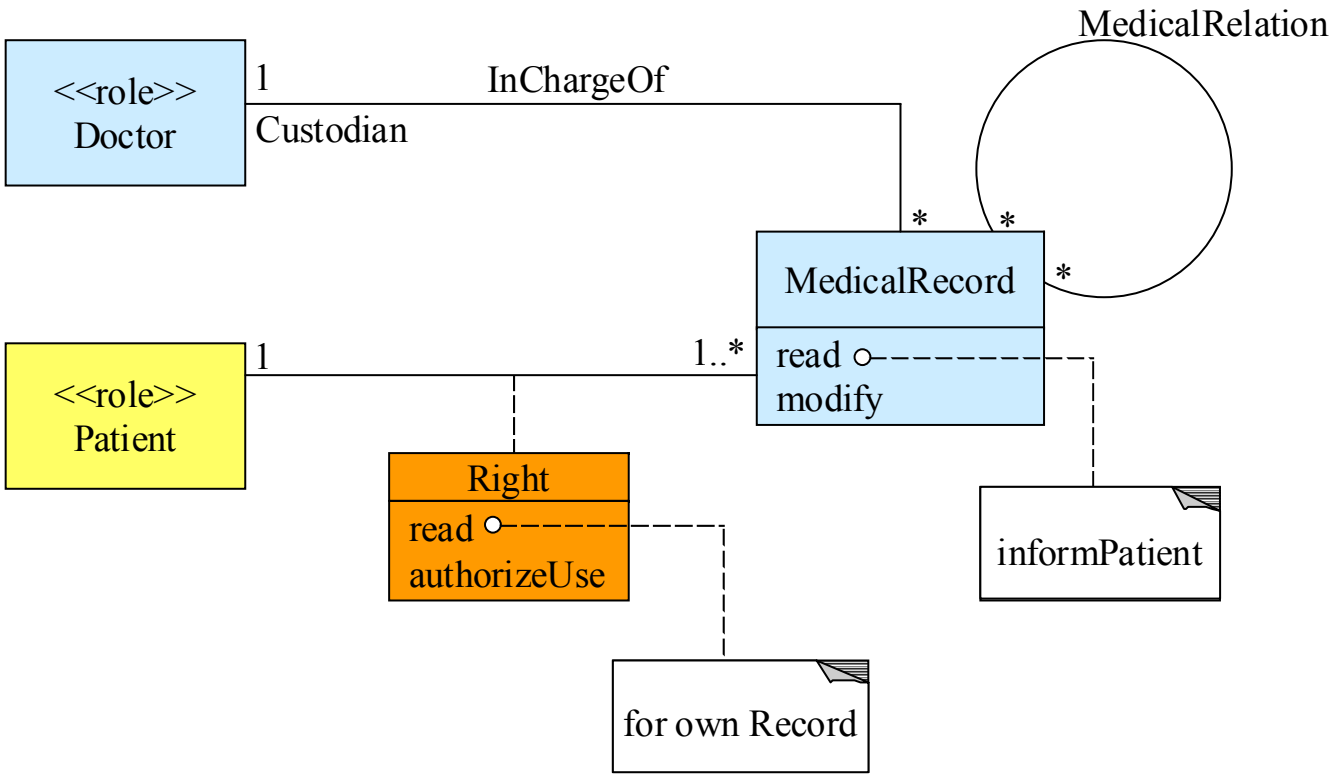



TreatmentInstance
date
financialNumber
*
MedicalHistory
insurance
dateRange
open()
create ( )
update ( )
1
Patient
name
address
patientNumber
create ( )
Right
treatPatient
readTreatmentInstanc
Right
admitPatient
assignAssets
Right
<<role>>
AdministrativeClerk
<<role>>
Nurse
<<role>>
Doctor
<<role>>
HospitalAuditor
Employee
name
idNumber
address
Hospital
name
address
MedicalGroup
name
Building
name
location
Location
number
size


Observ
AdmitPatientView
- newPatient
- initialComplaint
- patientNumber
-
patientInformation
- chart
- medicalHistory
- inpatient
AdmitPatientContr
oller
Mod
Patient
- name
- address
-
patientNumber
1
Right
treatPatient
1
<<role>>
AdministrativeC
lerk
MedicalHistory
- insurance
- dateRange
+ open()
+ create()
+ update()
Inpatien
t
Outpatie
nt
*
TreatmentInstance
- date
- financialNumber
- initialComplaint
+ create()
+ update()

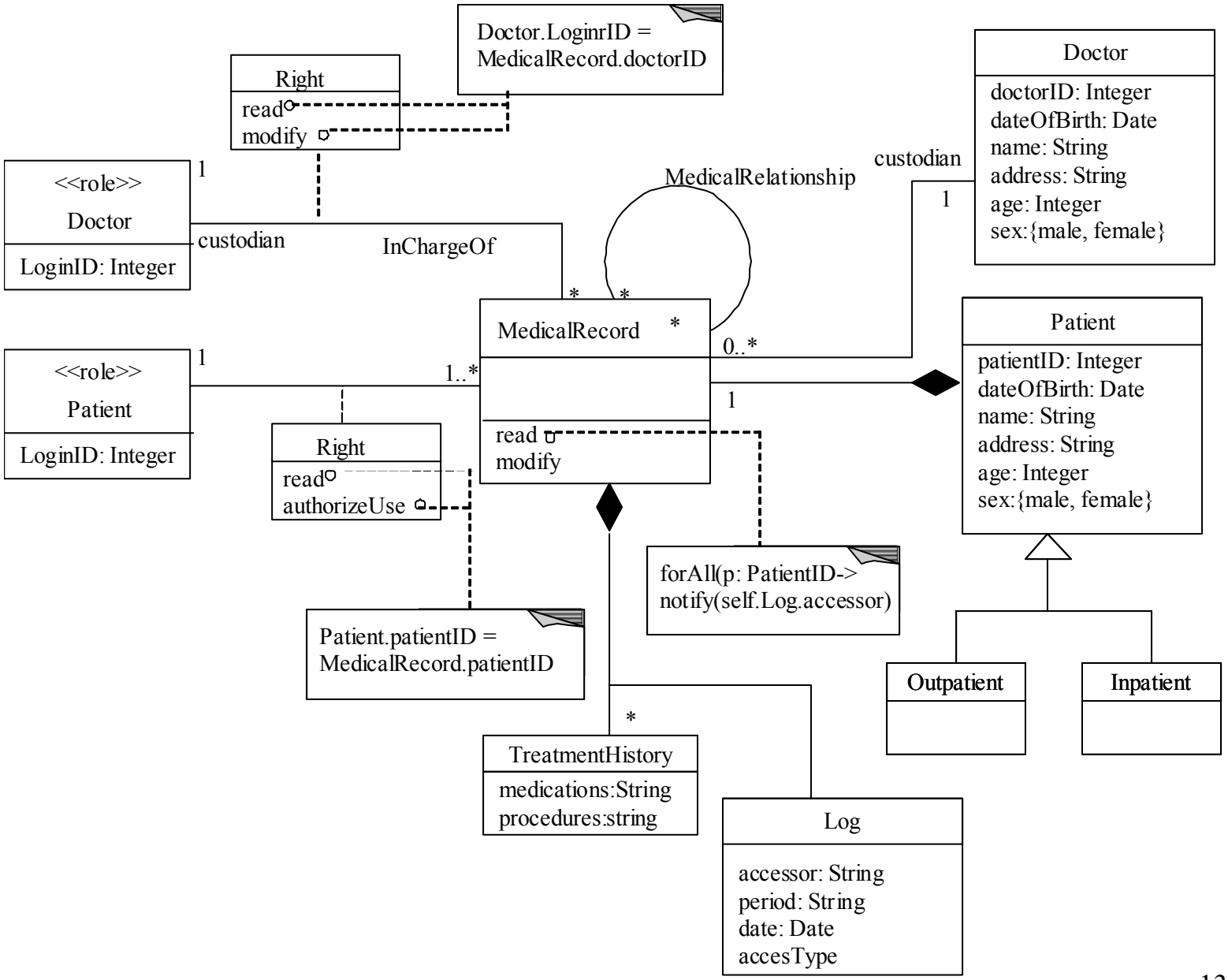
Some policies for medical information

- Patients can see their records, consent to their use, must be informed of their use
- A doctor or other medical employee is responsible for use of record (custodian)
- Records of patients with genetic or infectious diseases must be related


<<role>>
Doctor
1
InChargeOf
Custodian
MedicalRelation
*
*
*
MedicalRecord
read
modify
<<role>>
Patient
1
1..*
Right
read
authorizeUse
informPatient
for own Record

OCL (Object Constraint Language)

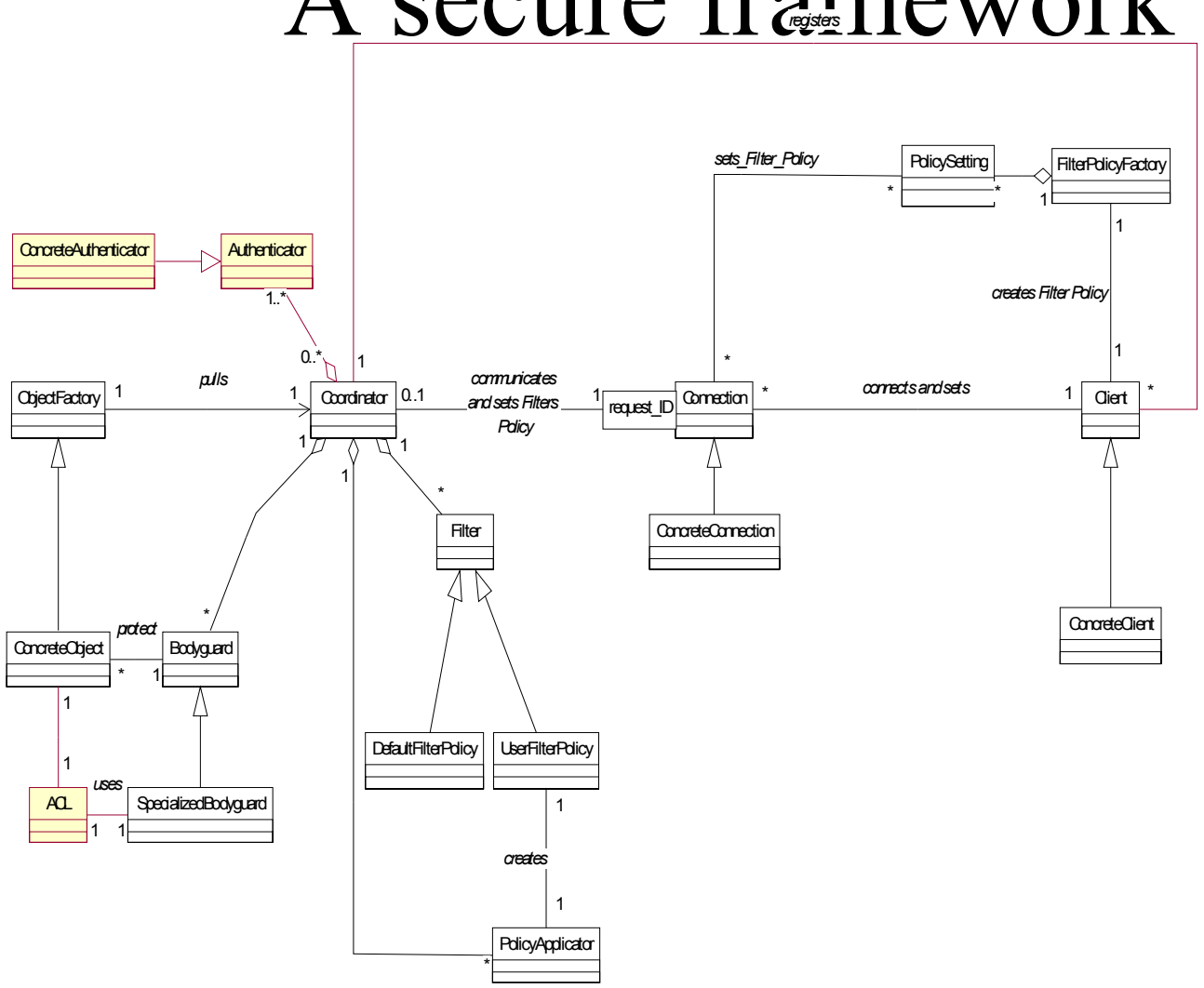
- Similar to Z and SQL, 1st order predicate calculus
- Adds precision to UML constraints
- Implementation oriented

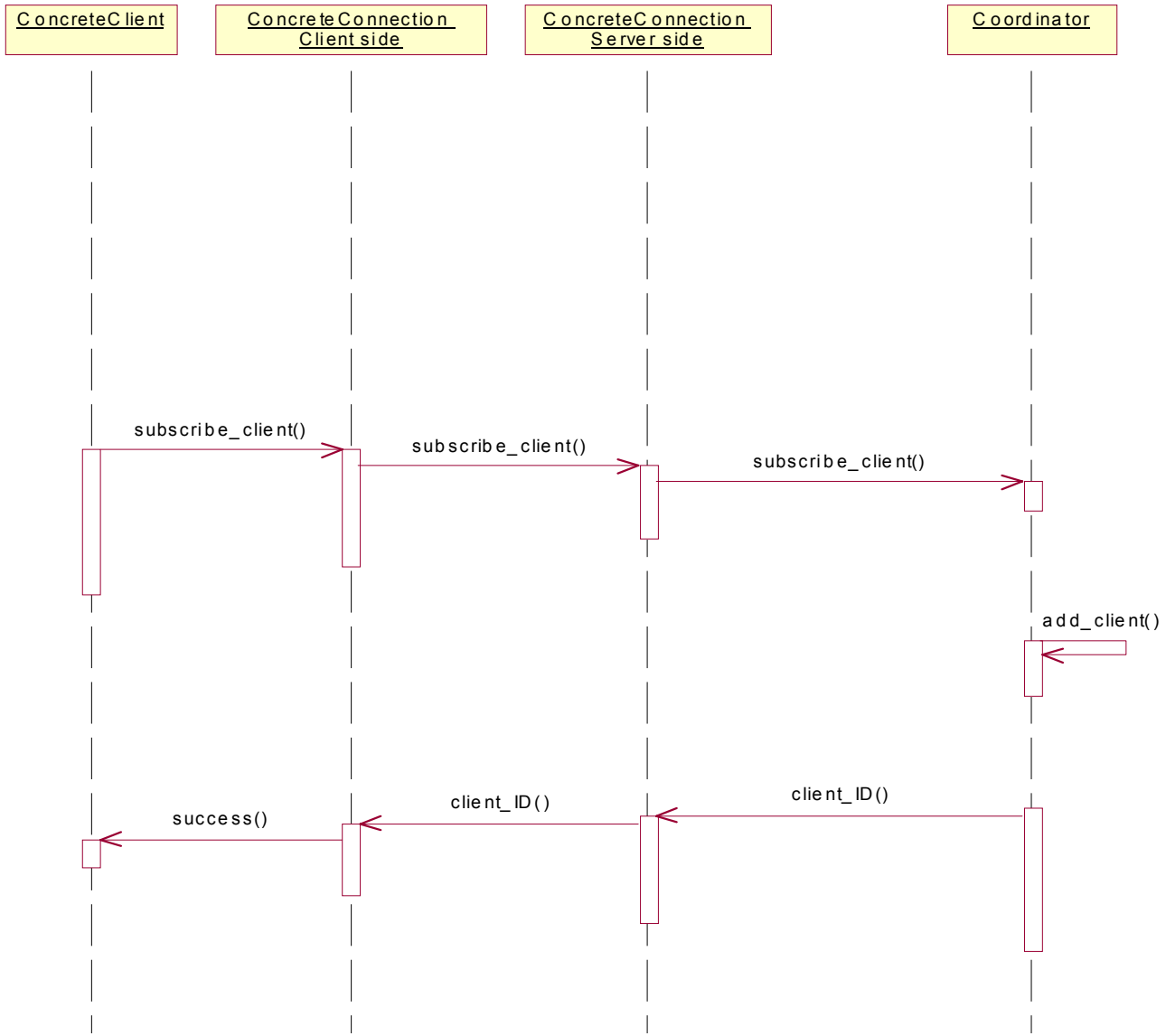

Doctor.LoginrID =
MedicalRecord.doctorID
Right
read
modify
Doctor
doctorID: Integer
dateOfBirth: Date
name: String
address: String
age: Integer
sex:{male, female}
<<role>>
Doctor
LoginID: Integer
1
custodian
InChargeOf
MedicalRelationship
custodian
1
*
*
MedicalRecord
*
0..*
<<role>>
Patient
LoginID: Integer
1
1..*
Patient
patientID: Integer
dateOfBirth: Date
name: String
address: String
age: Integer
sex:{male, female}
1
Right
read
authorizeUse
read
modify
forAll(p: PatientID->
notify(self.Log.accessor)
Patient.patientID =
MedicalRecord.patientID
Outpatient
Inpatient
*
TreatmentHistory
medications:String
procedures:string
Log
accessor: String
period: String
date: Date
accesType

Distributed systems and web applications

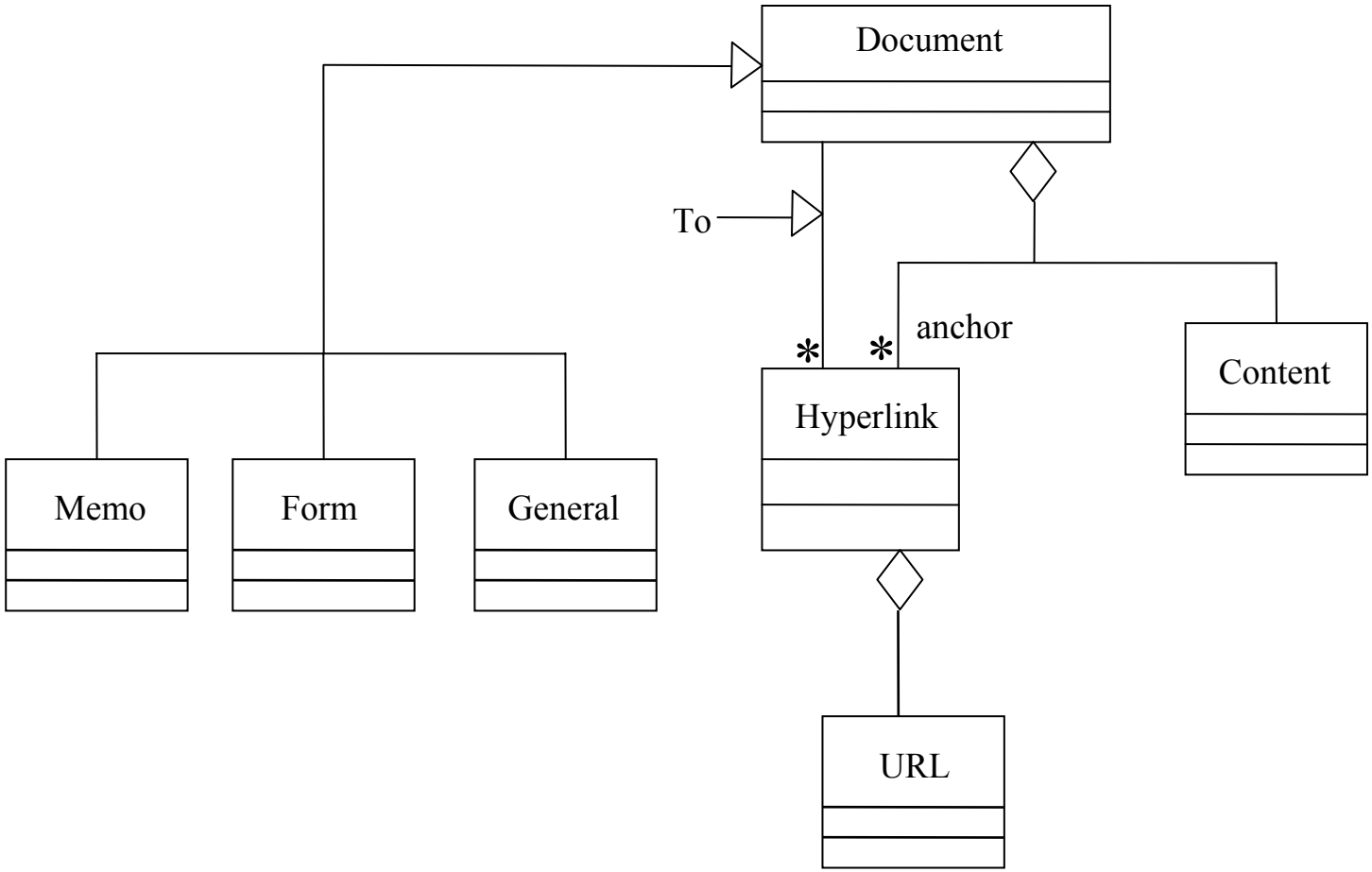
- Framework for filtering, access control, establish connection
- Shows combination of distribution and security patterns

A secure framework

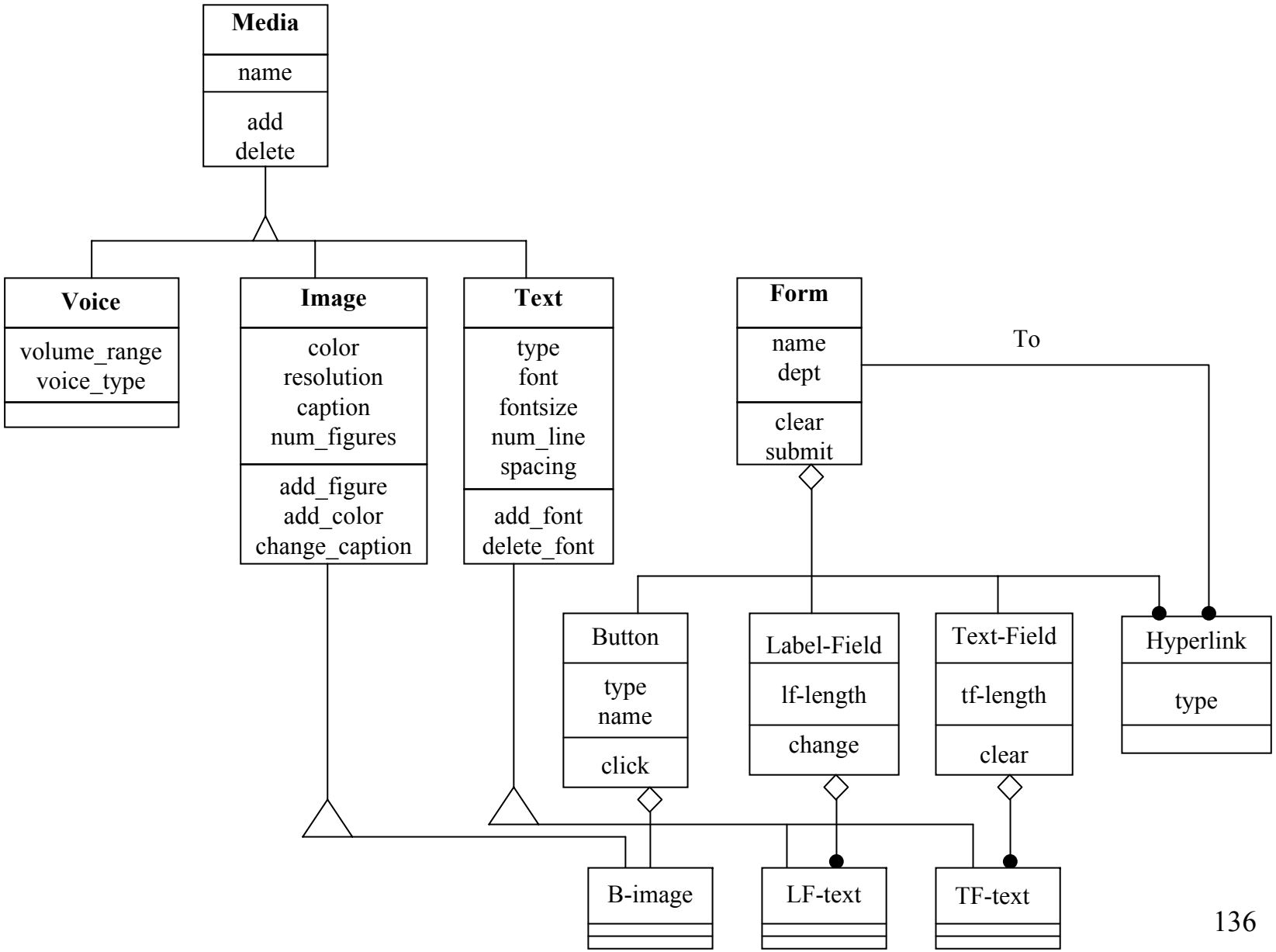



ConcreteClient
ConcreteConnection Client side
ConcreteConnection Server side
Coordinator
subscribe_client()
subscribe_client()
subscribe_client()
add_client()
client_ID()
client_ID()
success()

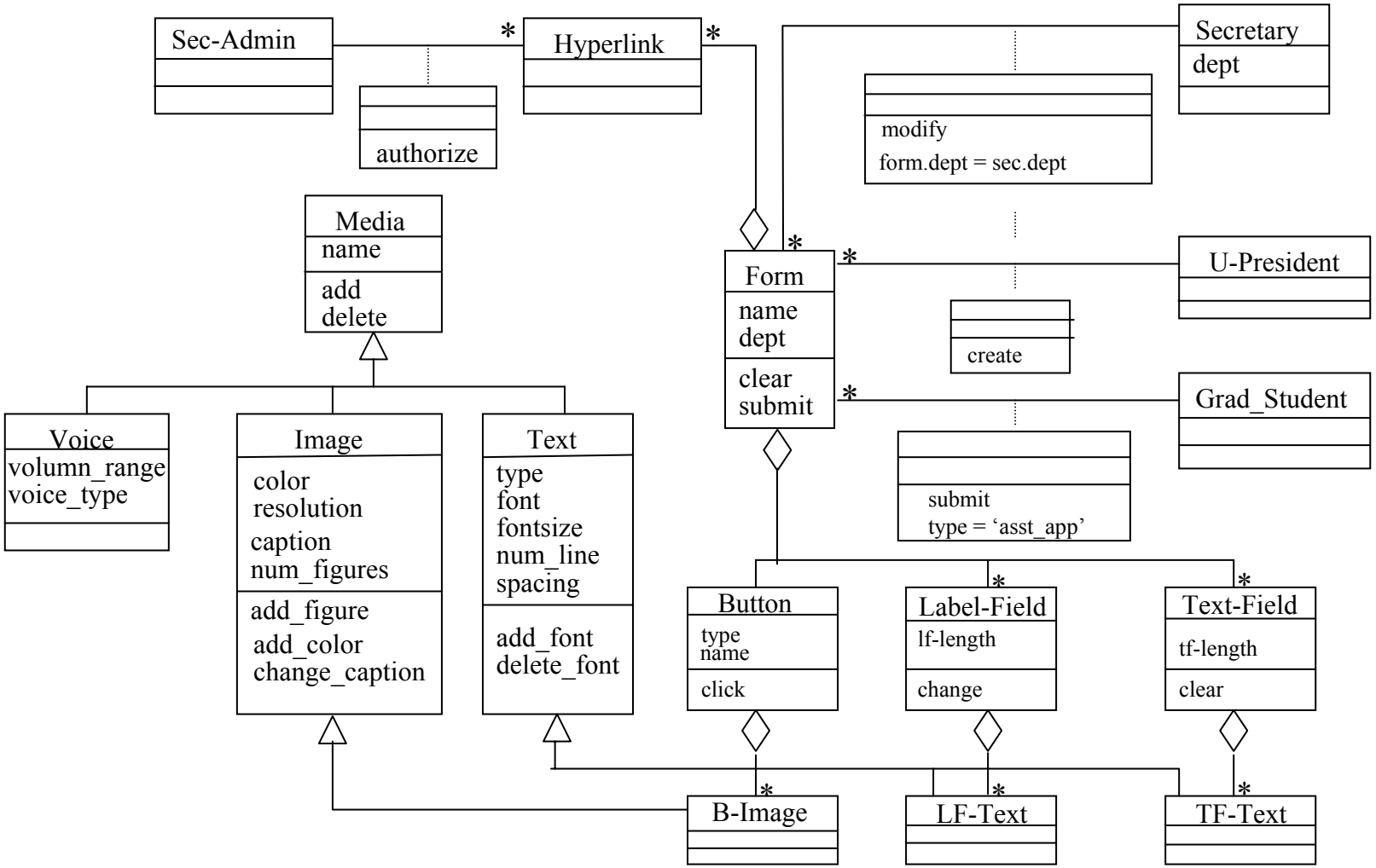
Web Documents



Example -- Form



OO models of web documents



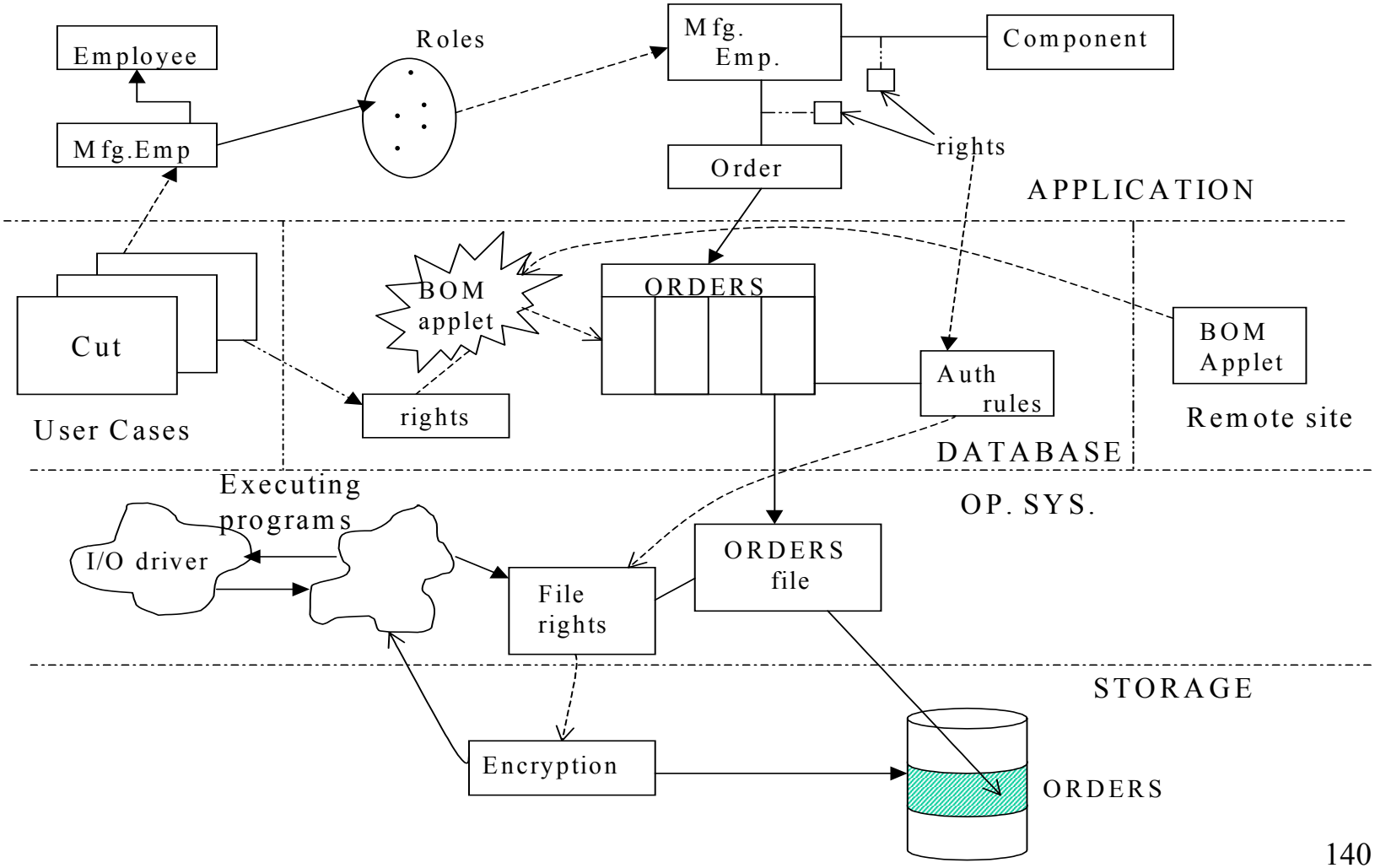
Other security patterns

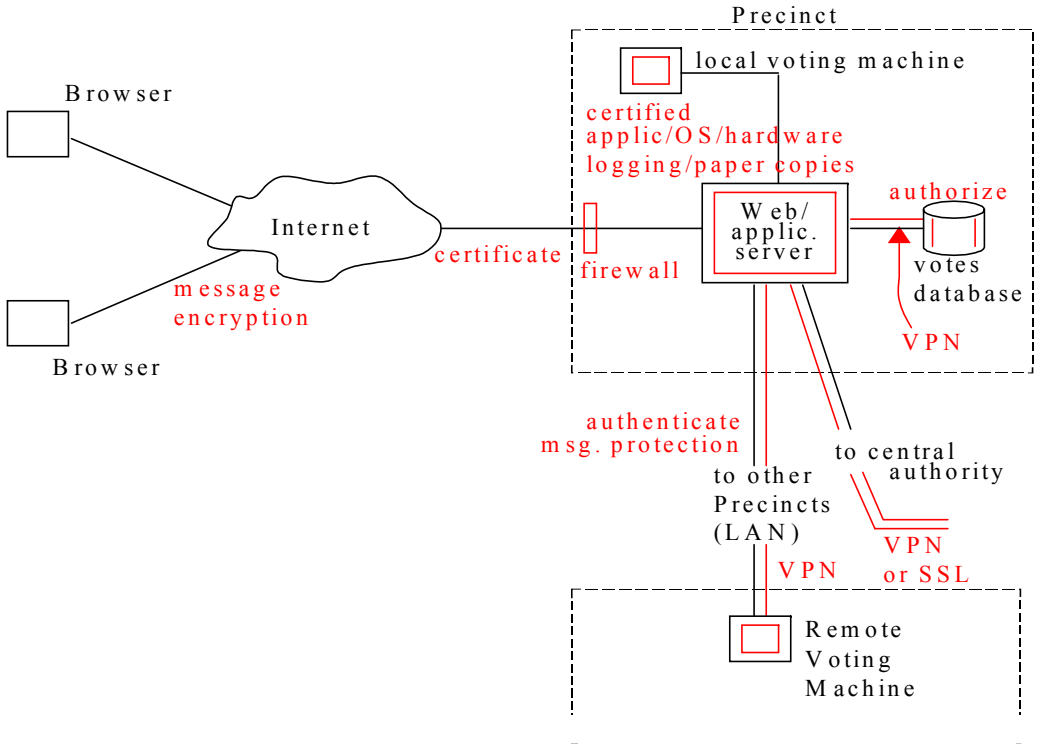
- Patterns for RBAC implementation
- Cryptographic patterns
- Java security patterns
- Single Point of Access (Joe Yoder)
- M. Schumacher, E.B.Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad (Eds.), *Security Patterns,* Wiley 2005 (to appear).

Secure architectures

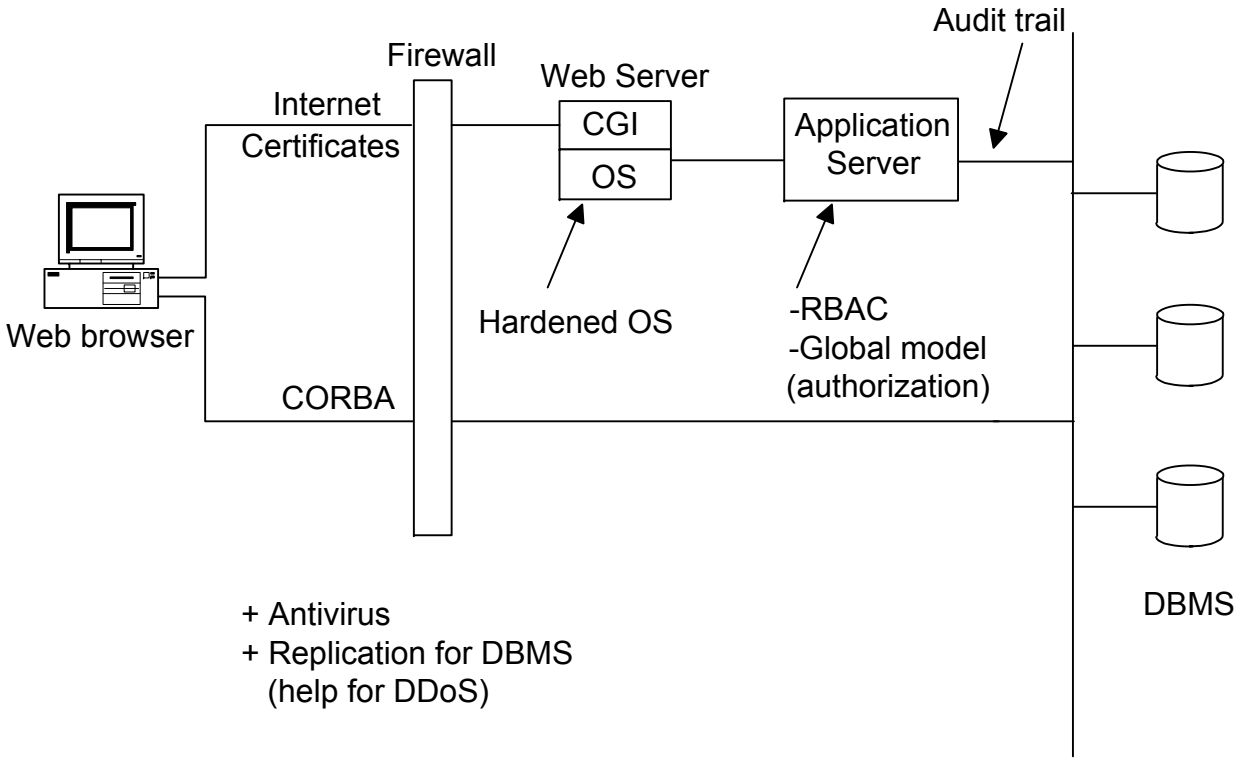
- Apply patterns at each level according to attacks
- Determine appropriate security mechanisms from patterns

Mapping of authorization rules




Precinct
local voting machine
Browser
certified
applic/OS/hardware
logging/paper copies
authorize
Internet
Web/
applic.
server
certificate
firewall
votes
database
message
encryption
Browser
VPN
authenticate
msg. protection
to central
authority
to other
Precincts
(LAN)
VPN
or SSL
VPN
Remote
Voting
Machine

(4) Implementation



Conclusions and future work

- Internet-based systems are very flexible, but also very complex and changing
- Currently security is rather poor
- We must design new systems or improve existing systems in a systematic way
- Proposed methodology is a good step to build secure systems

Future work

- Patterns for web services standards
- Patterns for database systems
- Build a systematic catalog of security patterns
- Define precise mappings between levels
- Refine the development method: components, distribution