Tutorial:
Security patterns and secure systems design using UML

Eduardo B. Fernandez and Maria M. Larrondo Petrie
Dept. of Computer Science and Eng.
Florida Atlantic University
www.cse.fau.edu/~security
{ed, maria}@cse.fau.edu

About Eduardo

- 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

About Maria

- Professor of Computer Science & Engineering and Associate Dean of Academic & International Affairs at Florida Atlantic University, Boca Raton, FL., USA
- Executive Vice President of LACCEI
  (Latin American and Caribbean Consortium of Engineering Institutions)
- Authored over 130 research papers on complex systems modeling (environmental, security)
- Over US$2.5M in research grants from NATO, IBM, NSF, and South Florida Water Management District

Abstract

- Analysis and design patterns are well established to build high-quality object-oriented software. Patterns combine experience and good practices to develop basic models that can be used for new designs. Security patterns join the extensive knowledge accumulated about security with the structure provided by patterns to provide guidelines for secure system design and evaluation. We show a variety of security patterns and their use in the construction of secure systems. These patterns include Authentication, Authorization, Role-based Access Control, Firewalls, Web Services Security, and others. We apply these patterns through a secure system development method based on a hierarchical architecture whose layers define the scope of each security mechanism. First, the possible attacks and the rights of the users are defined from the use cases using a Role-Based Access Control (RBAC) model. The attacks are used to find the necessary policies, while the rights are reflected in the conceptual class model. We then define additional security constraints that apply to distribution, interfaces, and components. The patterns are shown using UML models and some examples are taken from our book *Security Patterns: Integrating security and systems engineering* (Wiley 2006).

- **Keywords:** object-oriented design, patterns, secure systems design, security, software architecture

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

SECURITY PATTERNS

Integrating Security and Systems Engineering

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

Objectives

- To get a panorama of security patterns and how to use them
- To consider a systematic approach to secure systems development based on patterns and UML
- To study some specific patterns in detail
- To get ideas for research

Outline 1

- Security concepts and definitions
- The Internet
- Attacks
- A methodology for the design of secure systems
- Security models
- Firewalls
- Operating systems

Outline 2

- Web services
- Application security
- Distributed and web systems
- More security patterns
- Conclusions

Security

- Objectives
- Countermeasures
- Security architectures

- “Me da lo mismo que me escuches o no. Es asi, y me parece justo que lo sepas”.

- J. Cortazar, “El idolo de las Cicladas”, en “Ceremonias”, Seix Barral, 1983

The 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

Basic security architecture

- *Authentication* happens first
- *Authorization rules* define what is allowed or not allowed (who can see what and how)
- *Assurance* is a measure of how well the lower levels enforce authentication and authorization
- *Cryptography* protects communications and maybe stored data

Security environments

- Early systems were isolated and single user --few security problems
- Mainframes brought many users but we knew them (registered)—complexity and attacks increased
- The Internet opened up our systems to unknown users—exponential growth in attacks

The Internet

- Basic architecture
- Documents
- New architectures

Al abrir los ojos, vi el Aleph. El lugar donde estan, sin confundirse, todos los lugares del orbe, vistos desde todos los angulos.

J.L. Borges, “Narraciones”, Salvat, 1982.

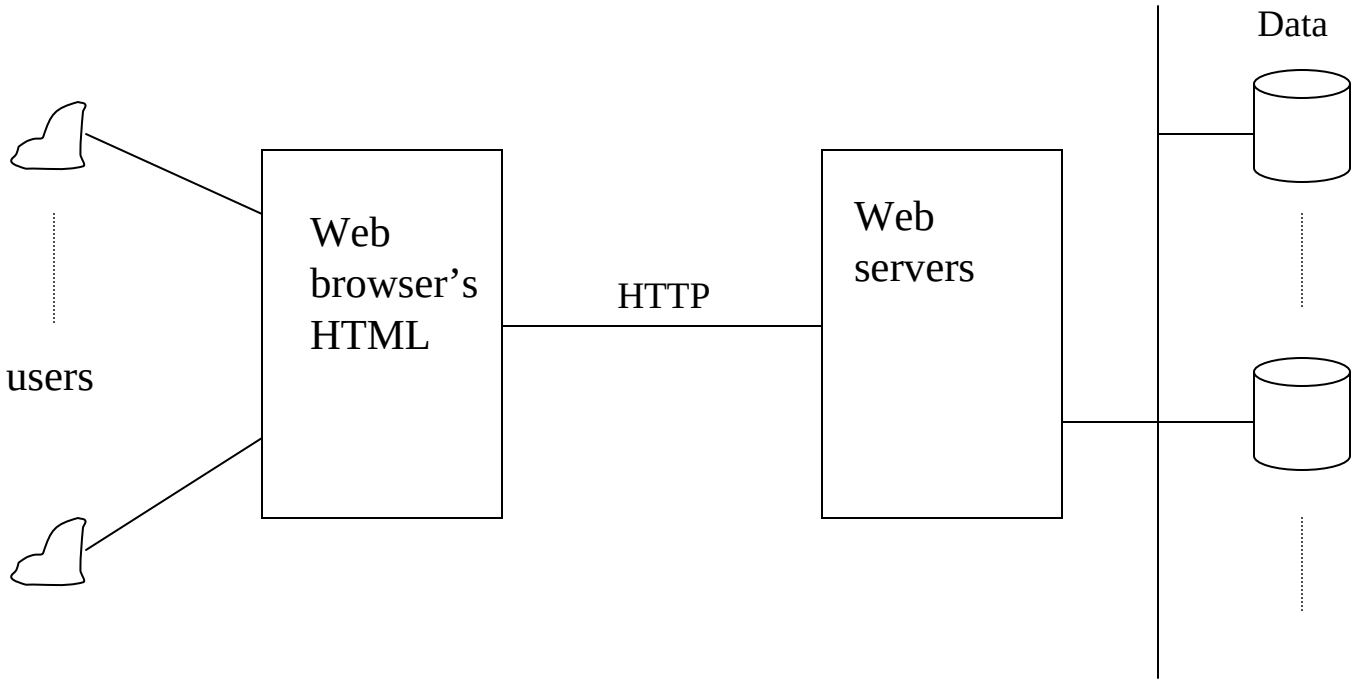
Context for security

- The Internet is the most demanding environment
- Complex
- Unknown users
- We want to have our information accessible from the Internet

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

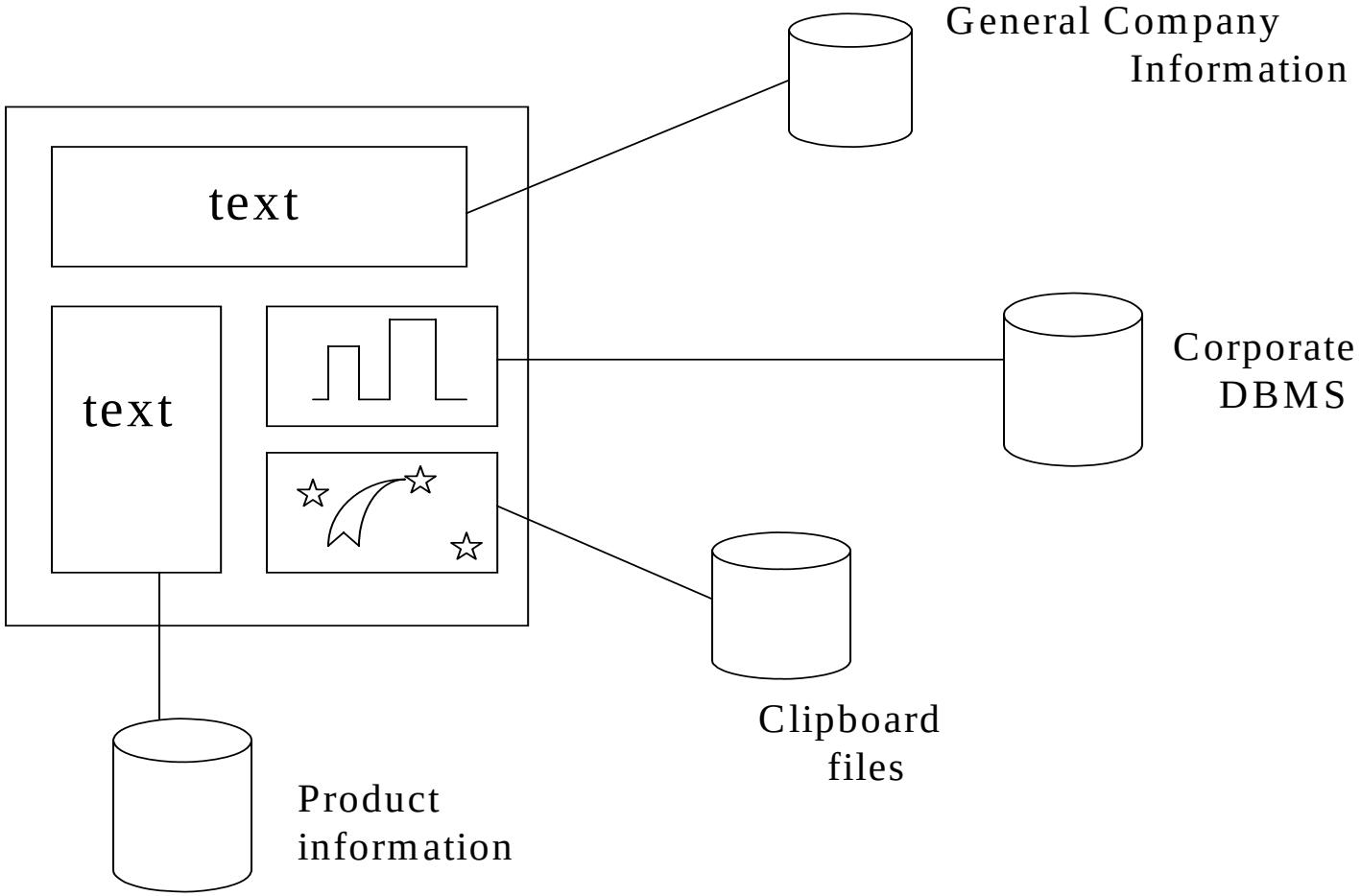
Basic Internet architecture



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

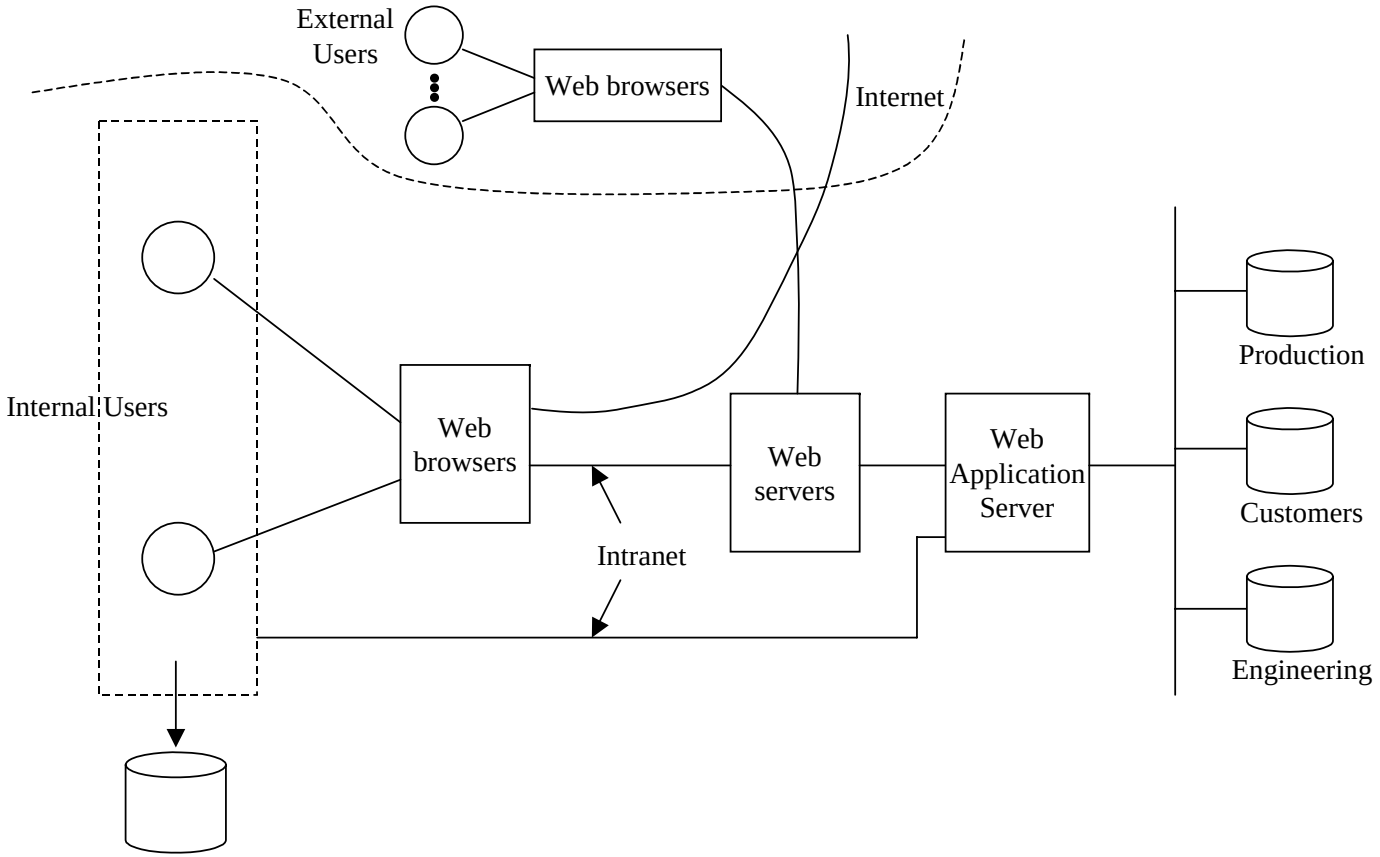
Example of a web page



XML

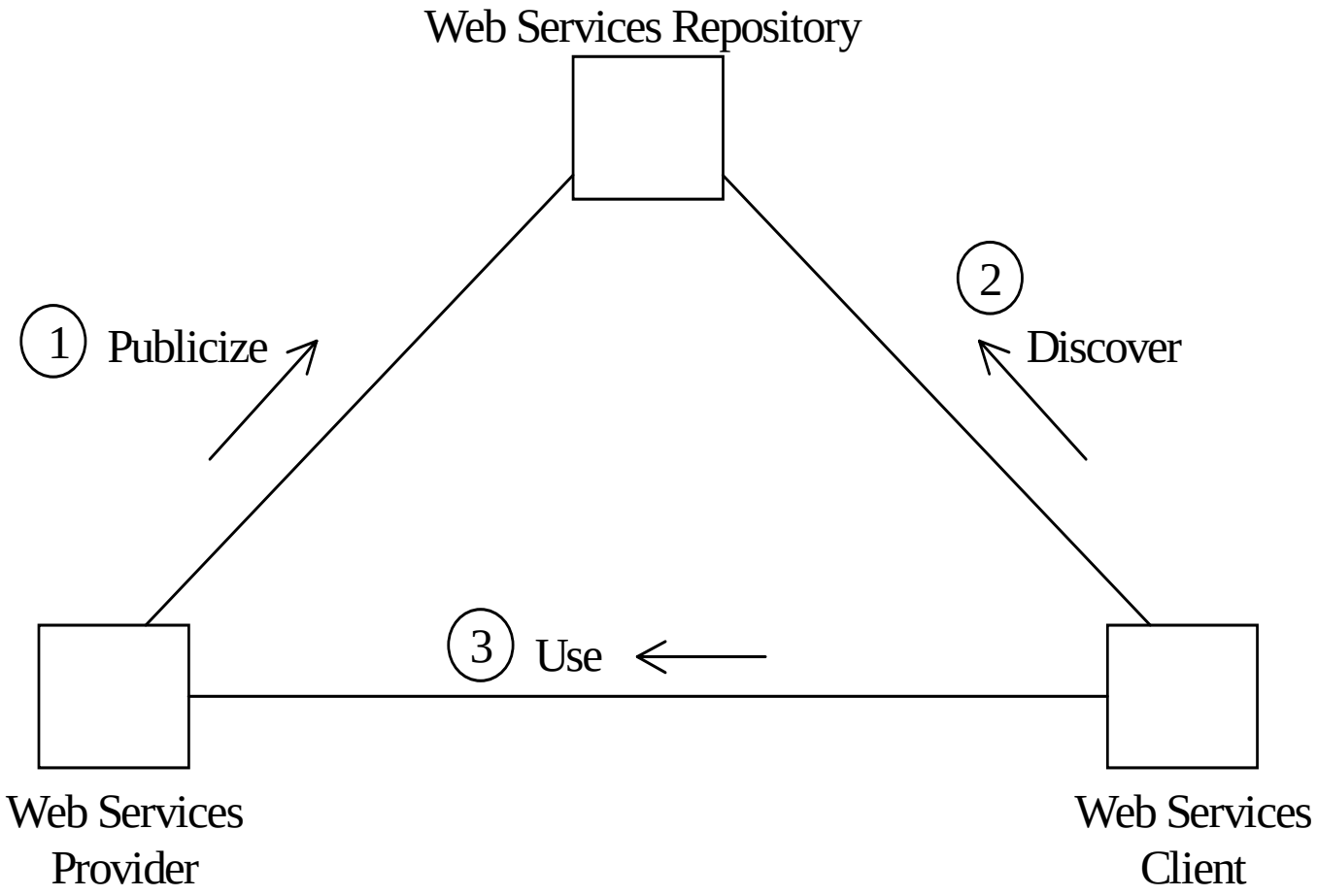
- XML is a metalanguage to define the meaning and structure of documents. A subset of SGML (Standard Generalized Markup Language). Basic ideas: use tags in data items to define their meaning, relate data items through nesting and references.

Enterprise architectures



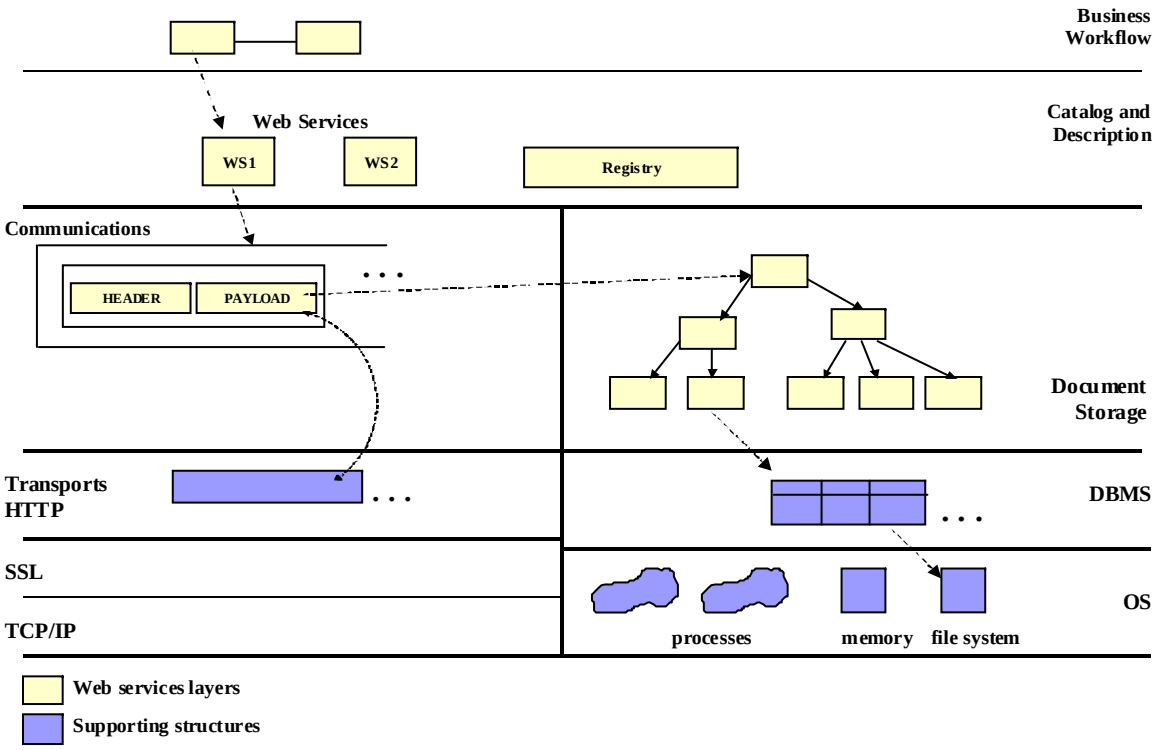
Web Services

- 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
- Requires a standard supporting framework
- The web could become a marketplace of web services (not there yet)


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

Web services architectures

- Web services are 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

Agents

- Autonomous software that moves through the Internet
- Can perform predefined tasks, e.g. search for a book and buy it if the price is right
- No general standards until now

Attacks

- Methods
- Types

Pero a veces me encontraba perdido en la oscuridad o tenia la impresion de enemigos escondidos…Quienes eran esas gentes y que querian?

E. Sabato, “El tunel” (Seix Barral, 1978, p. 58)

Malicious code (malware)

- *Trojan Horses*—A Trojan Horse is an apparently useful program that has harmful hidden functions (spyware)
- *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.

Direct attacks

- To the operating system
- To the database system
- To the application (increasing)
- Done through the network
- Almost no attacks to the messages in the network (low payoff and cryptography works)

Attackers

- Insiders -- According to studies about half of the attacks to a system come from insiders.
- Hackers -- Usually try to show off their ability by penetrating systems
- Spies -- Industrial or government spionage

Vulnerabilities

- Attacks can exploit vulnerabilities to misuse information
- A threat is a potential attack
- An exploit or incident is a specific occurrence of an attack
- Complexity brings along more vulnerabilities

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 systems and application software
- We need a systematic way to build secure software

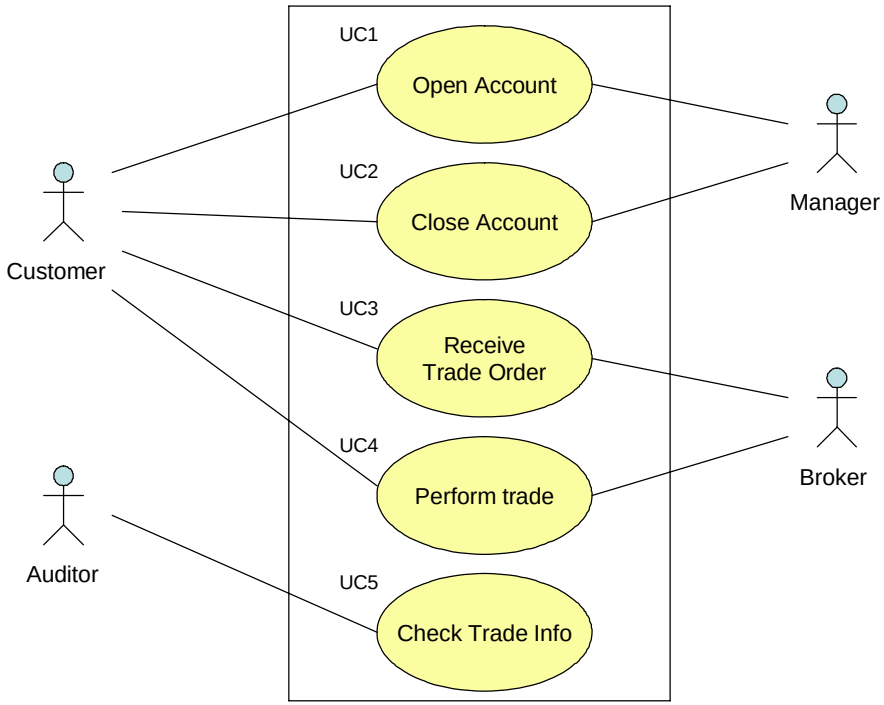
Identifying attacks

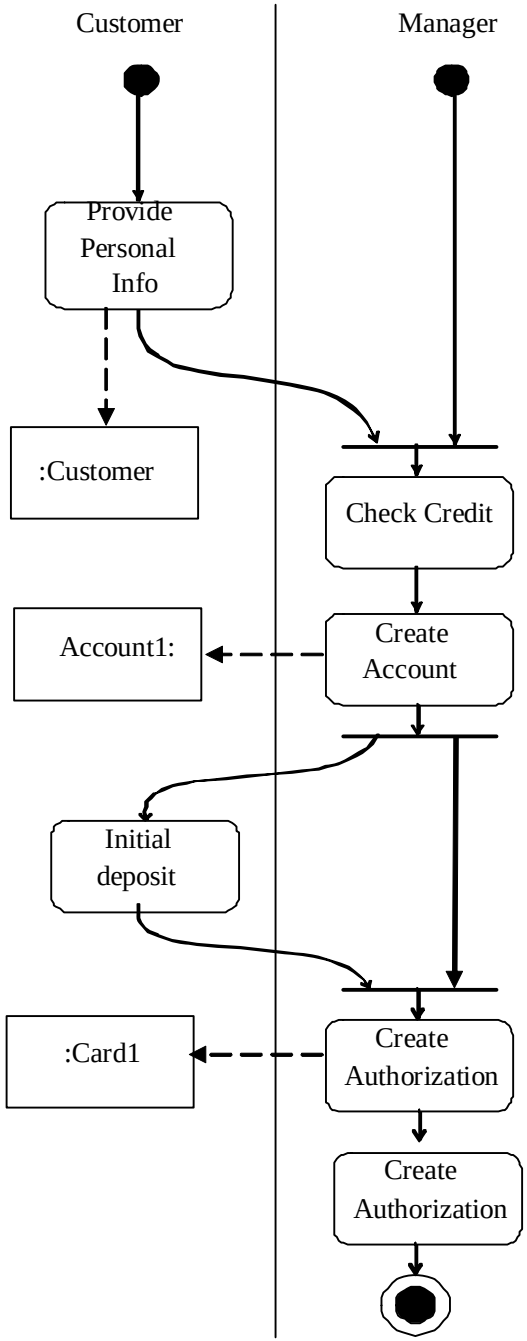
- We need to know what kind of attacks to expect.
- We relate attacks to attacker goals
- We study systematically all the possible attacks to each activity in a use case
- Use cases define all functional interactions

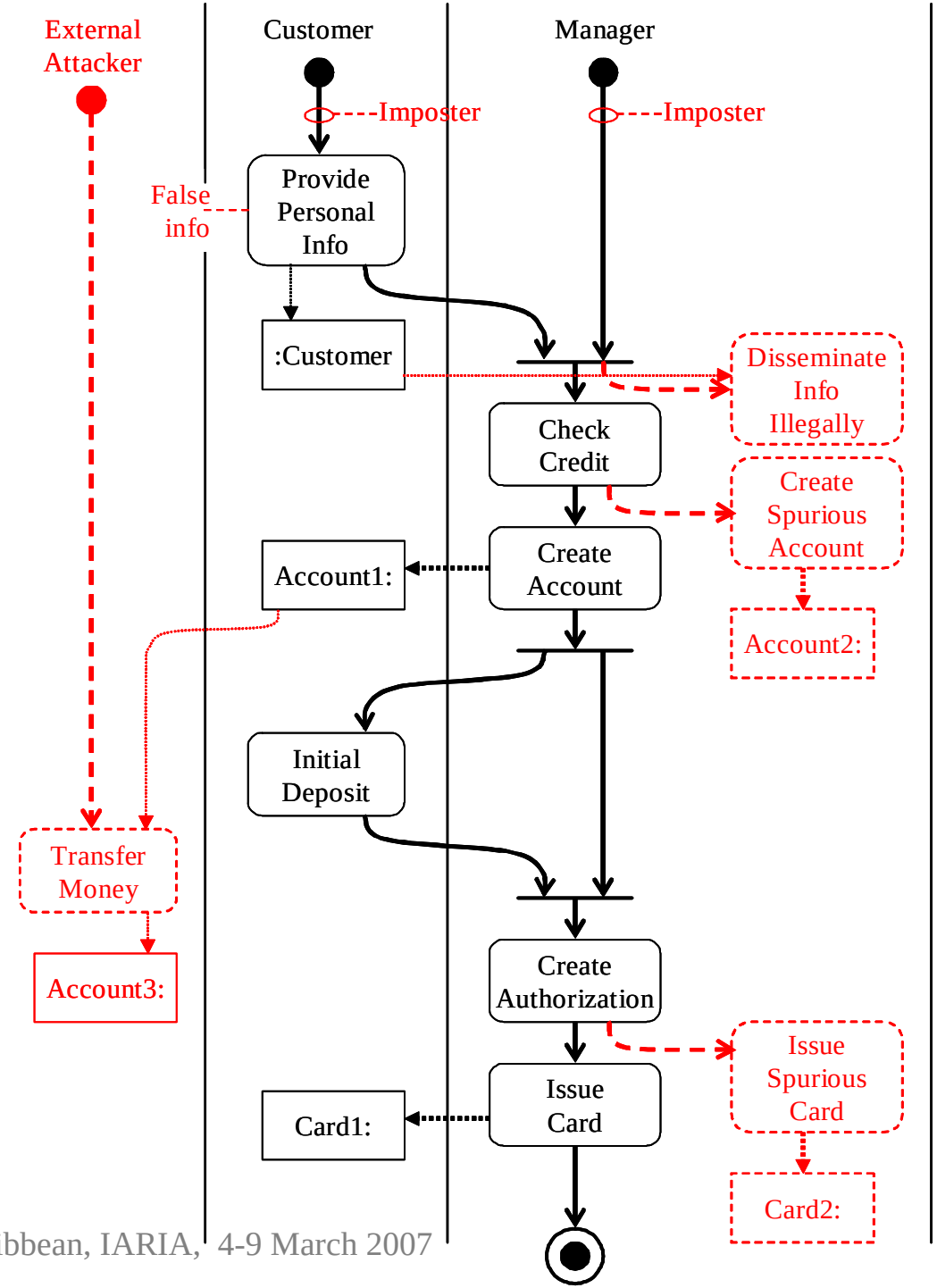
Use cases as starting point

- 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

A financial institution




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


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

Possible attacks

- A1.The customer is an impostor and opens an account in the name of another person
- A2.The customer provides false information and opens an spurious account
- A3.The manager is an impostor and collects data illegally
- A4.The manager collects customer information to use illegally
- A5.The manager creates a spurious account with the customer's information
- A6.The manager creates a spurious authorization card to access the account
- A7.An attacker tries to prevent the customers to access their accounts
- A8.An attacker tries to move money from an account to her own account

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

Trabajo desde hace an~os en la Unesco y otros organismos internacionales, pese a lo cual conservo algun sentido del humor y especialmente una notable capacidad de abstraccion, es decir que si no me gusta un tipo lo borro del mapa con solo decidirlo. De la misma manera si me gusta una chica puedo abstraerle la ropa apenas entra en mi campo visual,…

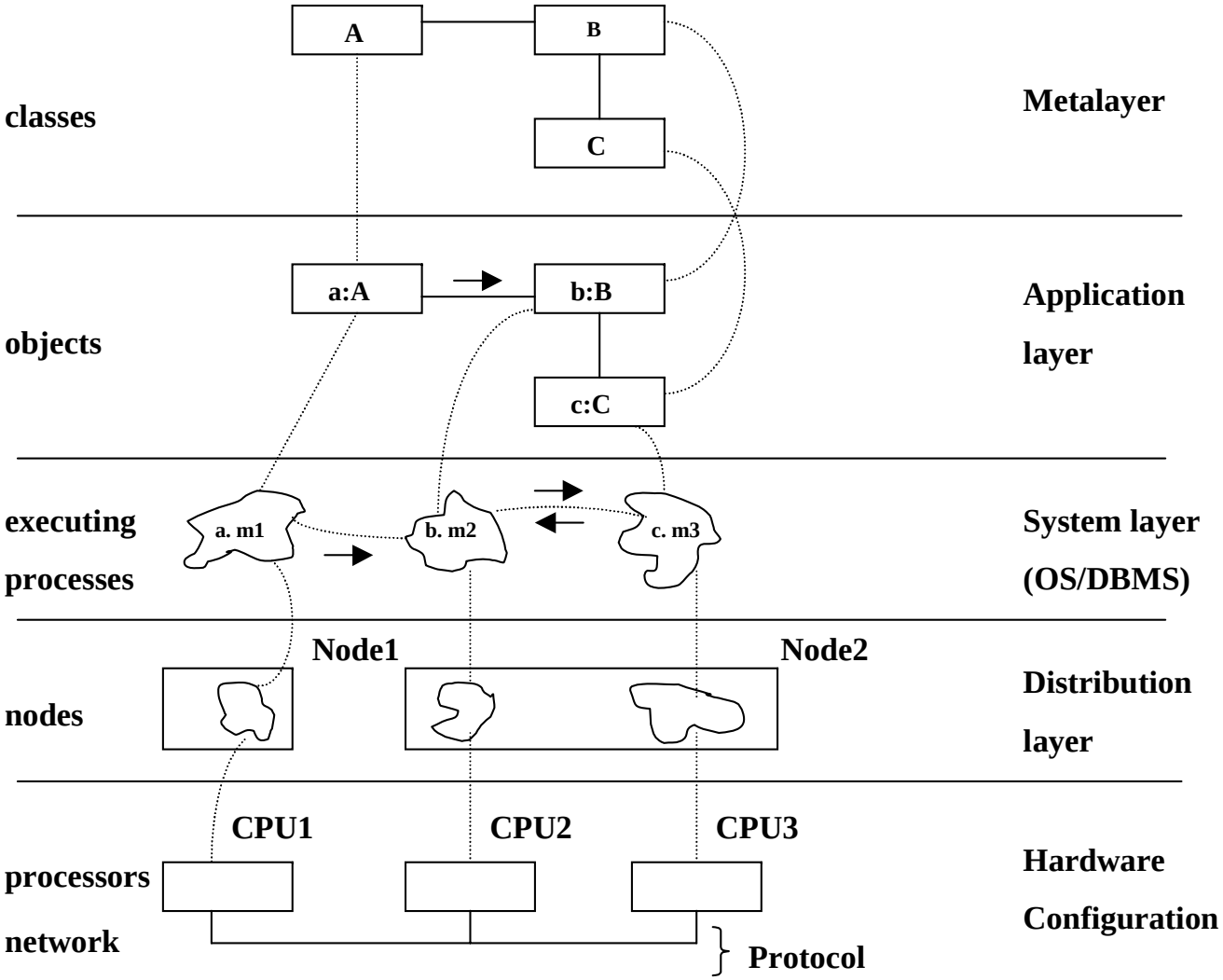
Julio Cortazar, "Historias de cronopios y de famas", Edhasa, 1970

Approach attempted in the past

- 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, valid for specific parts

Applying the principles

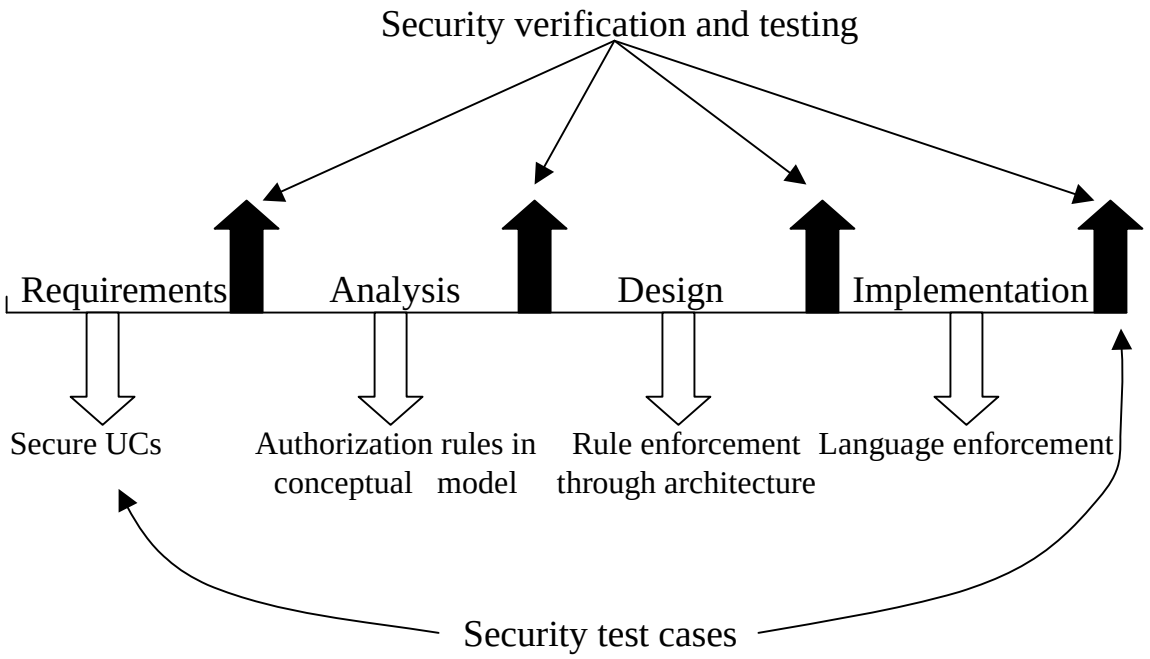
- Security should start where the application semantics is understood
- Security is an all-levels problem
- We should start from high-level policies and map them to the lower levels
- We need precise models to guide system development


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

Secure systems development methodology

- Apply security principles throughout the whole software lifecycle
- Use of object-oriented design
- Use cases identify attacks and 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.
- OMG standard (www.omg.org)
- Known and maybe used by many developers

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
- Can apply principles implicitly

Security patterns

- Analysis and design patterns are well established
- There are many principles of good design that have been developed to build secure systems
- It is possible to develop a collection of patterns that can be used to build secure systems
- Patterns can be used to build or evaluate secure systems or for teaching security

We can use patterns at all levels

- Patterns for models define the highest level
- At each lower level we refine the patterns at the previous level to consider the specific aspects of each level

We start from policies

- The policies of an institution define its way of accomplishing its objectives
- Security policies define its way to protect its information
- Without policies we don’t know what we should protect

Institution policies

- Laws, rules, and practices that regulate how an institution manages and protects resources. Another definition is: high-level guidelines concerning information security. Computer mechanisms should enforce these policies.

Some security policies

- Open/closed systems--In a closed system everything is forbidden unless explicitly allowed
- Need-to-know (Least privilege)-- Give enough rights to perform duties
- Information belongs to the institution versus private ownership
- Authorization-- access types, small units of access

Security policies II

- Obligation—What has to be done before accessing data
- Separation of duty—Separate critical functions into parts to be done by different people or systems
- Content-dependent access control—Access decision are based on the values of the data
- Authenticate all transactions—needed for accountability and access control

Example of university policies

- An instructor can look at all the information about the course he is teaching.
- An instructor can change the grades of the students in the course he is teaching
- A student may look at her grades in a course she is taking
- The department head can add/delete course offerings
- The registrar can add/delete students from course offerings
- Faculty members can look at information about themselves

Use of policies

- Secure systems must be closed but sometimes open access to information is more important, e.g., libraries, data warehouses, …
- The need-to-know principle must be applied with an appropriate granularity, many attacks happen because of too many rights

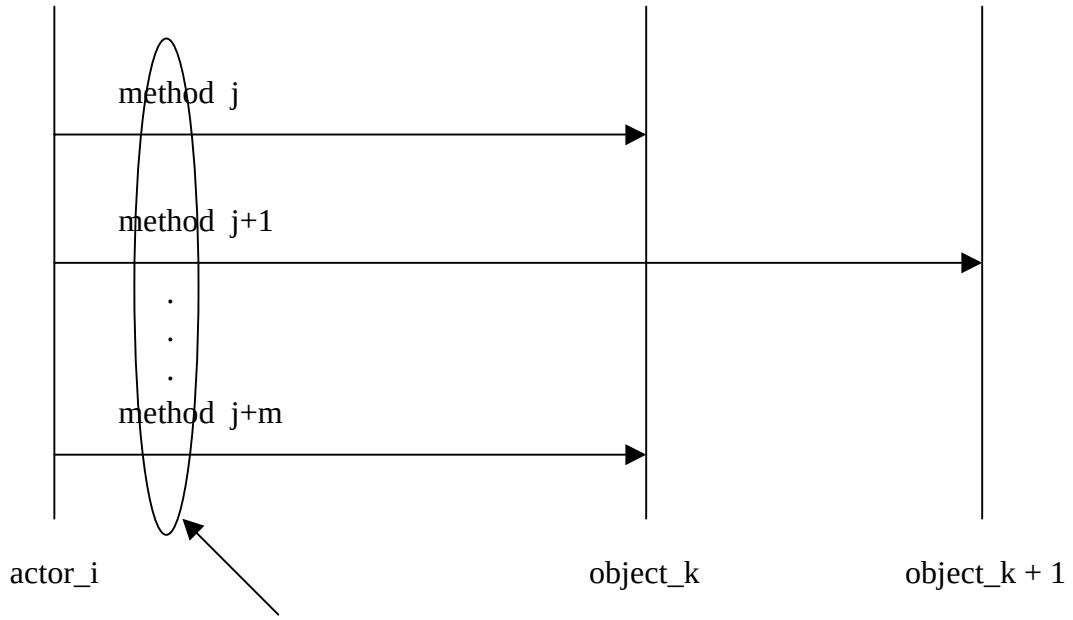
Use case analysis leads to policies

- A1. A3. Mutual authentication. Every interaction across system nodes is authenticated.
- A2. Verify source of information.
- A4. Logging. Since the manager is using his legitimate rights we can only log his actions for auditing at a later time.
- A5. A6. 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.
- A7. Protection against denial of service. We need some redundancy in the system to increase its availability.
- A8. Authorization. If the user is not explicitly authorized he should not be able to move money from any account.

Use cases can also be used to find actor rights (policies)

- Use cases describe all possible uses of the system
- All use cases define all possible and legal accesses
- Each actor can be given its needed rights to perform its functions

Scenarios to determine rights



Role rights for financial institution

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

Methodology

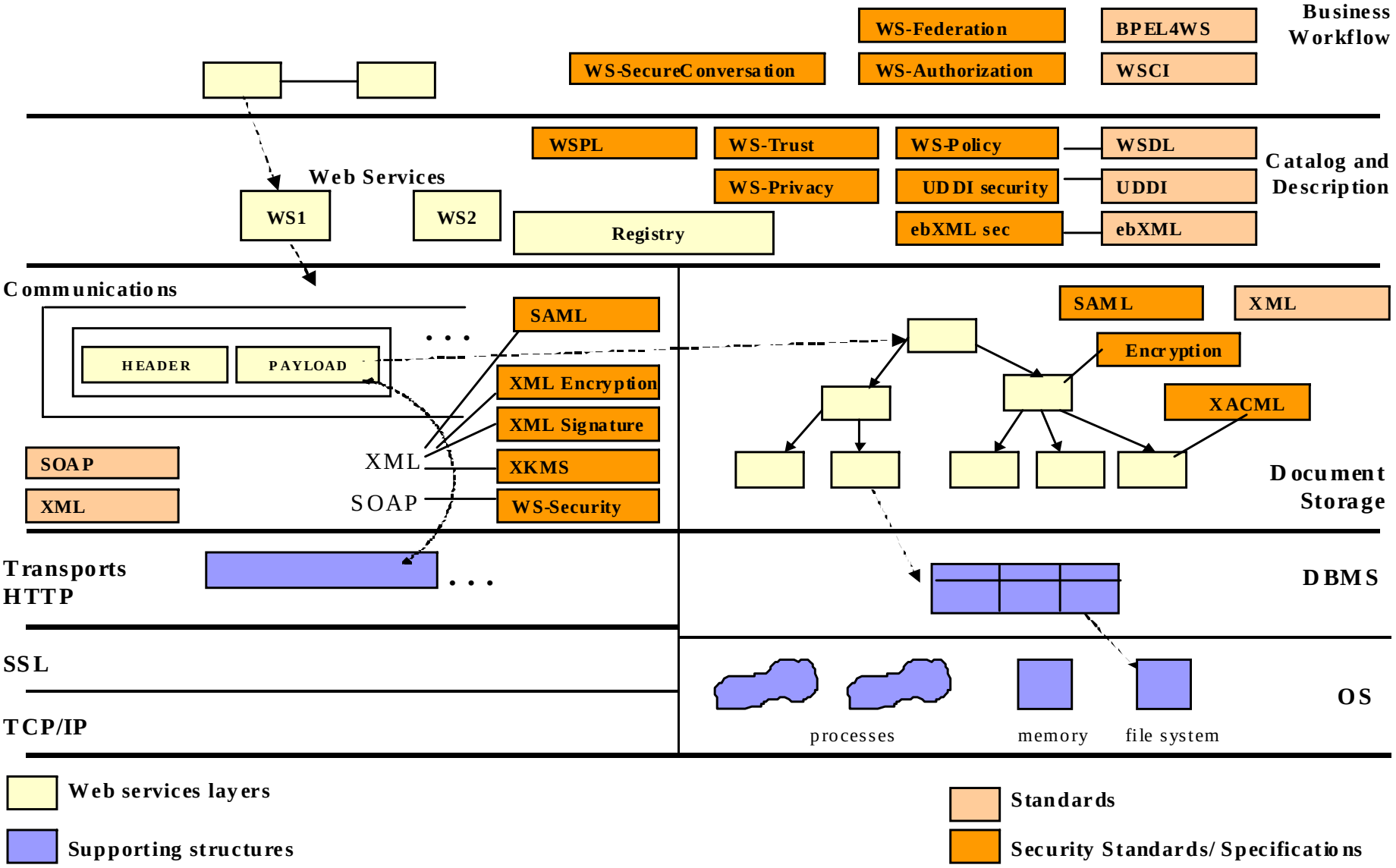
- Use case activities define attacks
- Attacks lead to policies to stop them
- Use cases define needed actor rights
- Access matrix or RBAC models formalize these rights

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


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

Security models

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

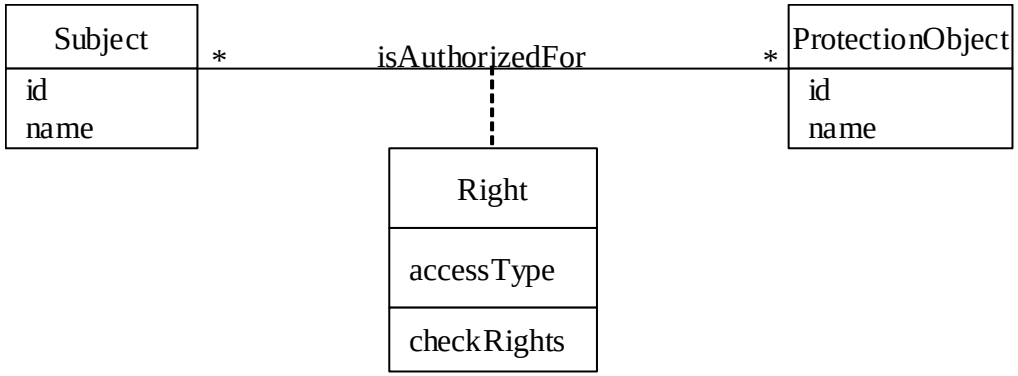
Classification of security models

- 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

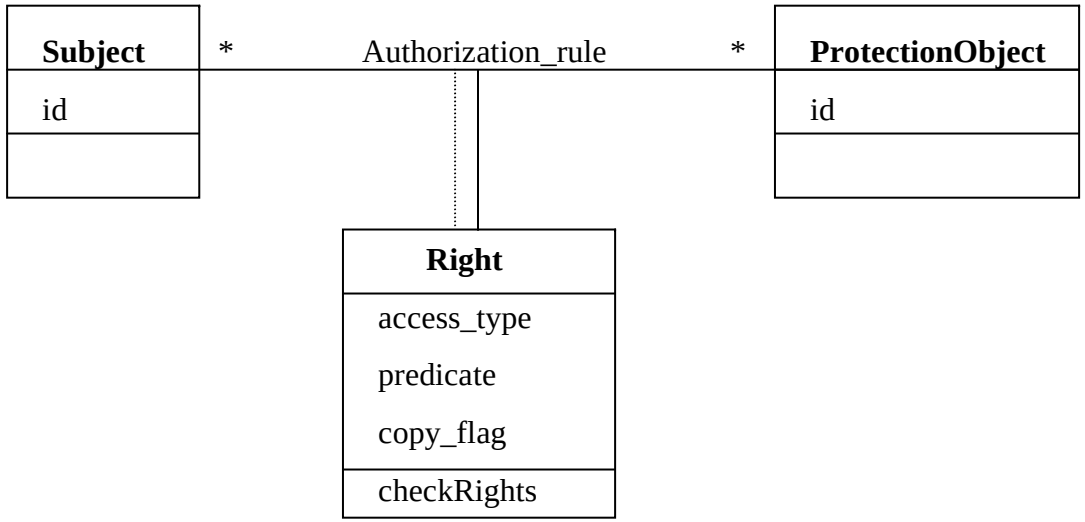
Access matrix authorization rules

- 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 patterns

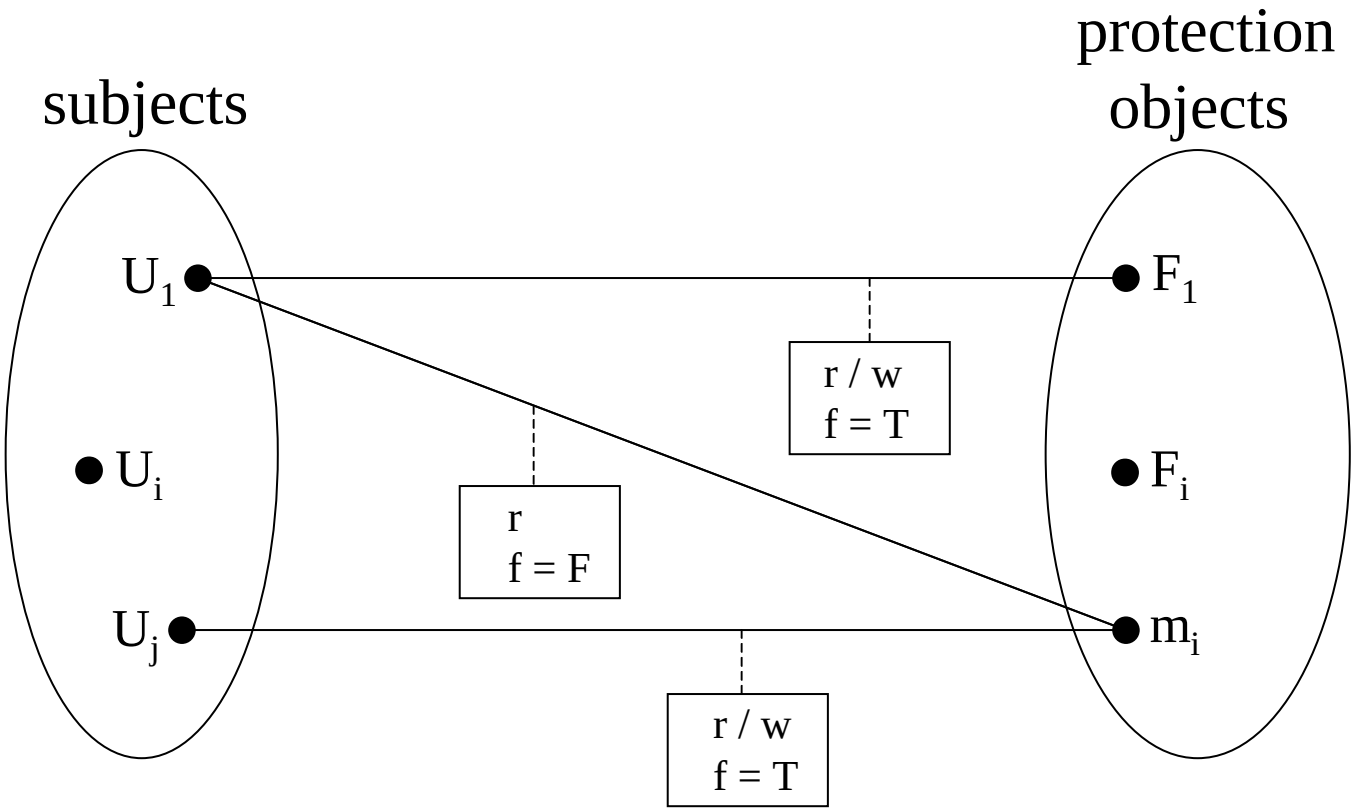
Authorization/access matrix



Extended access matrix



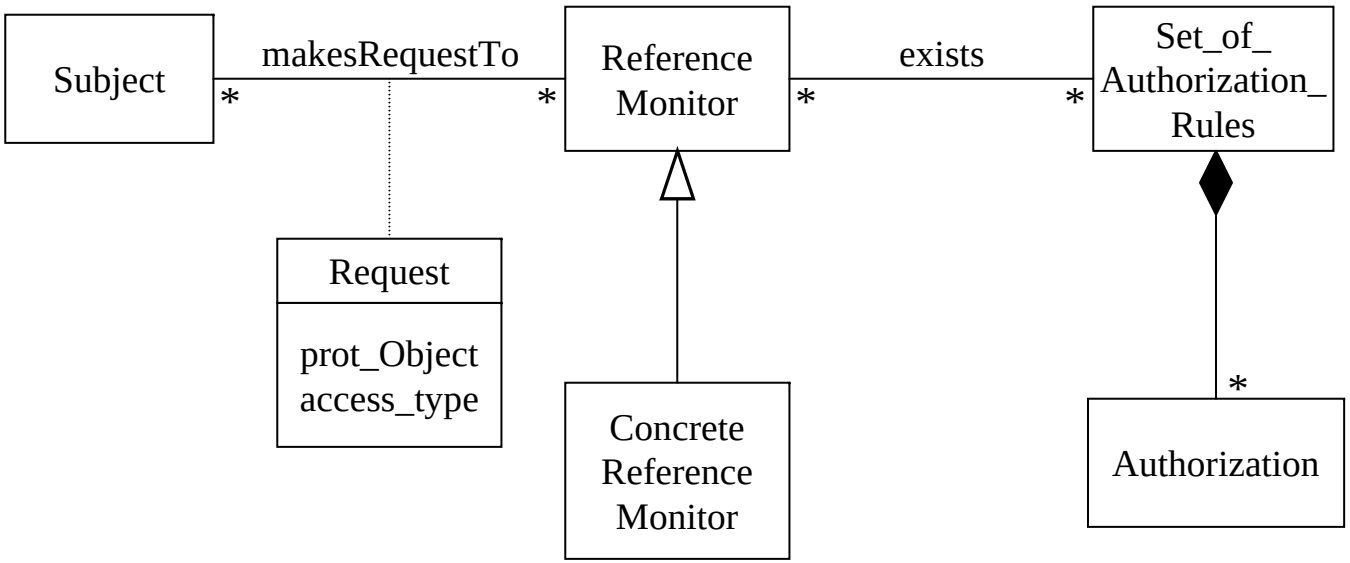
Authorization mapping



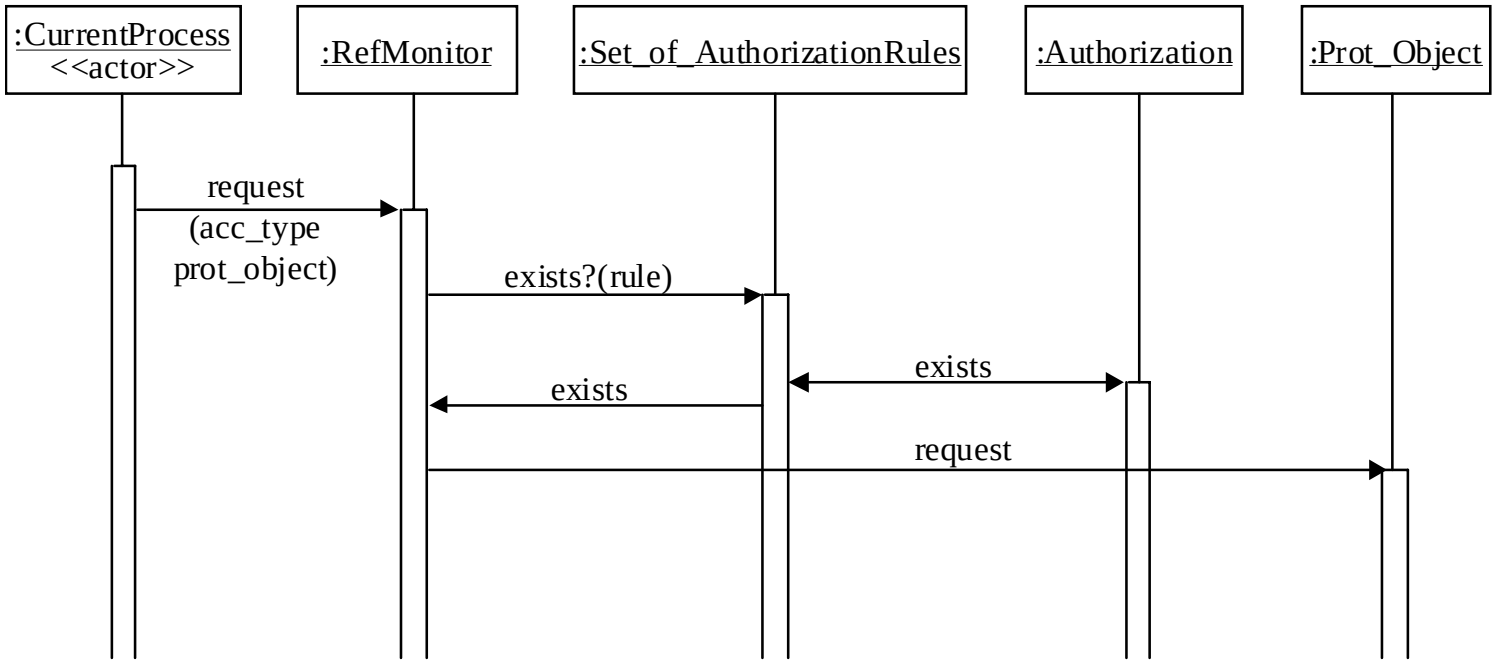
Reference Monitor

- 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 pattern



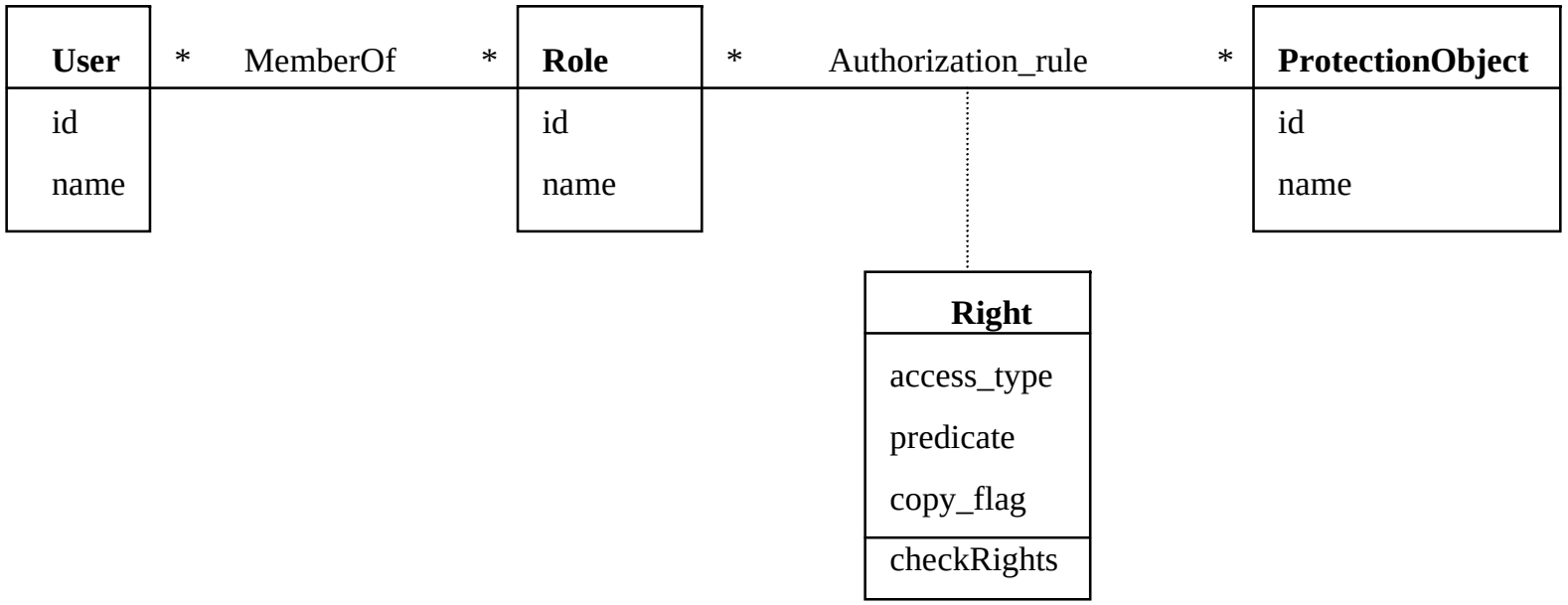
Enforcing access control



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

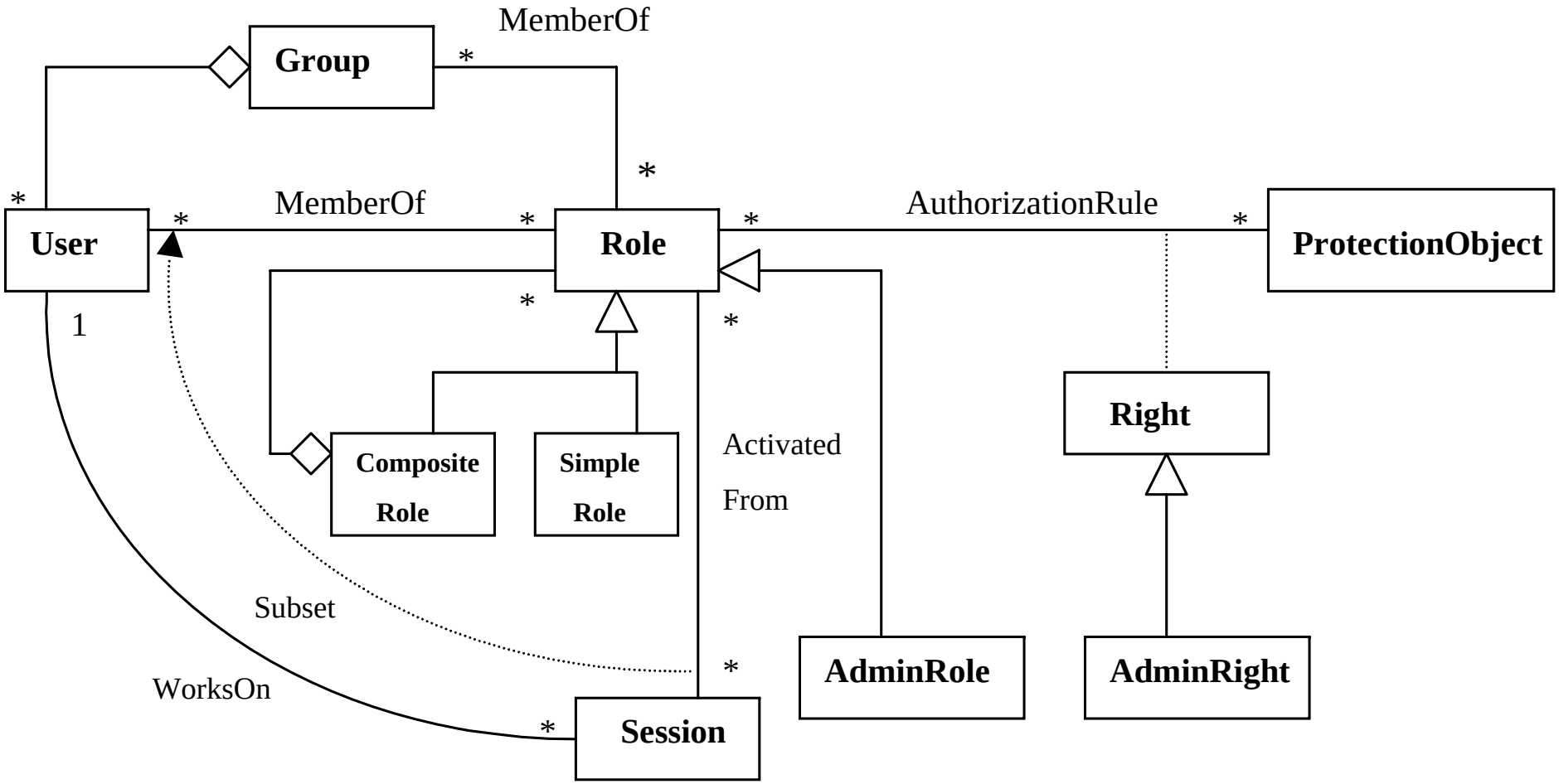
Basic RBAC pattern



Extended RBAC

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

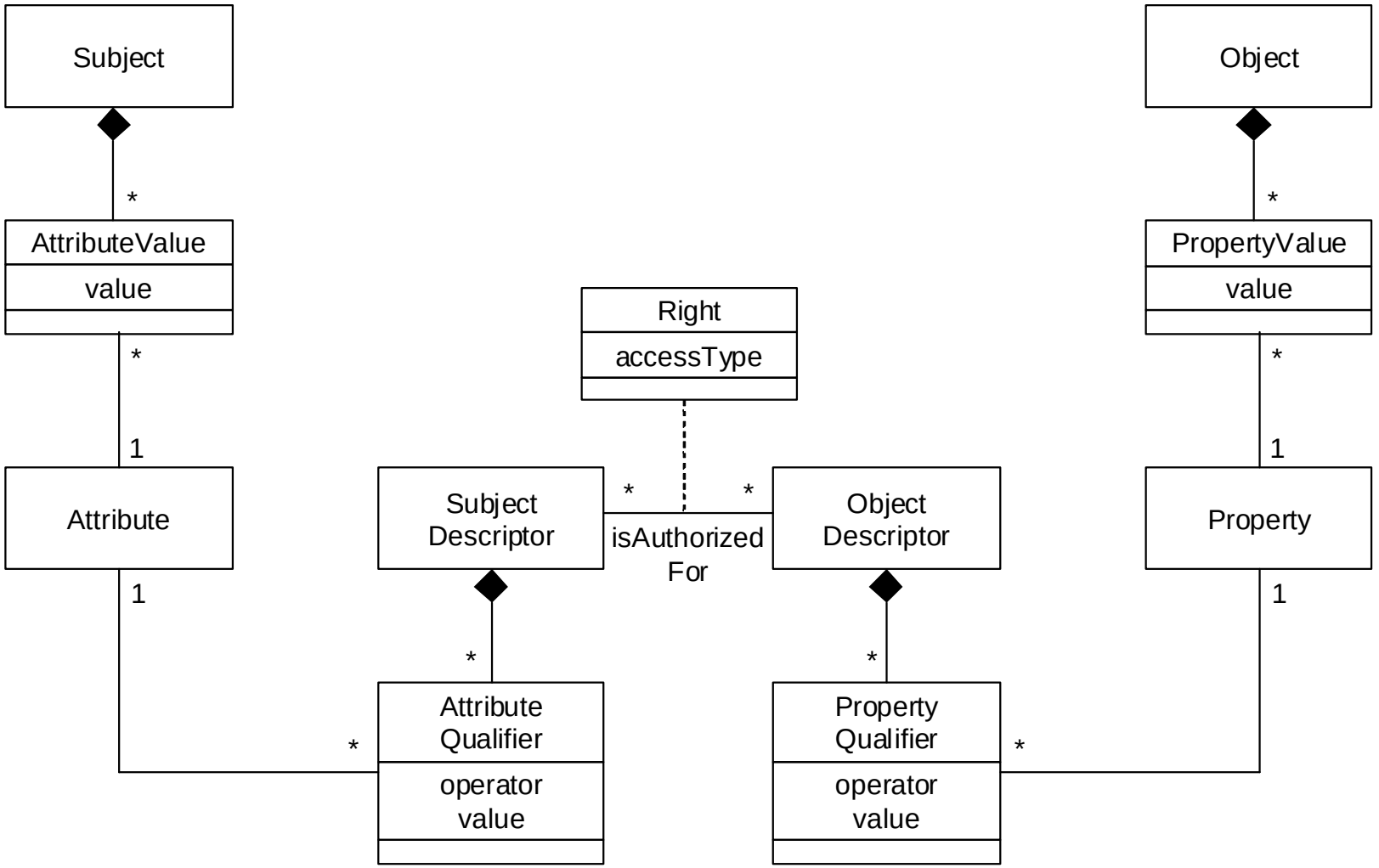
Extended RBAC pattern



Attribute-Based Access Control

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

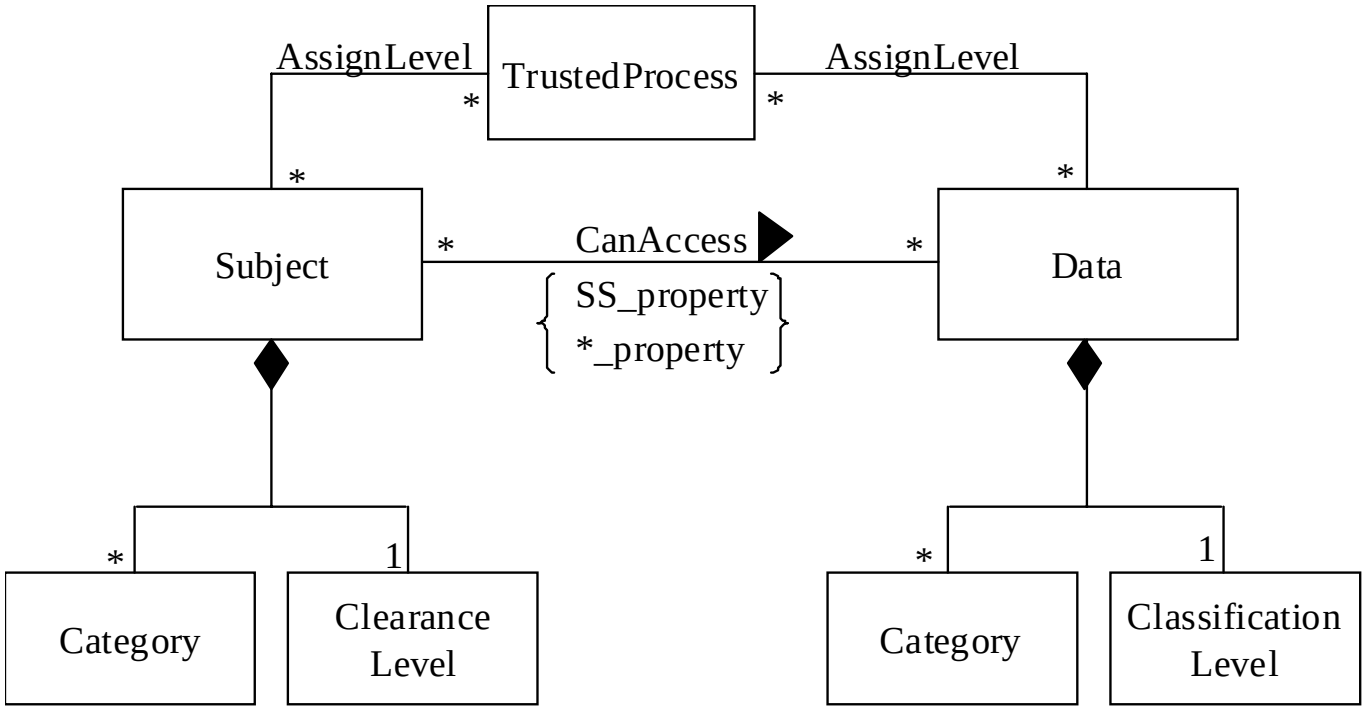
Metadata-based access control



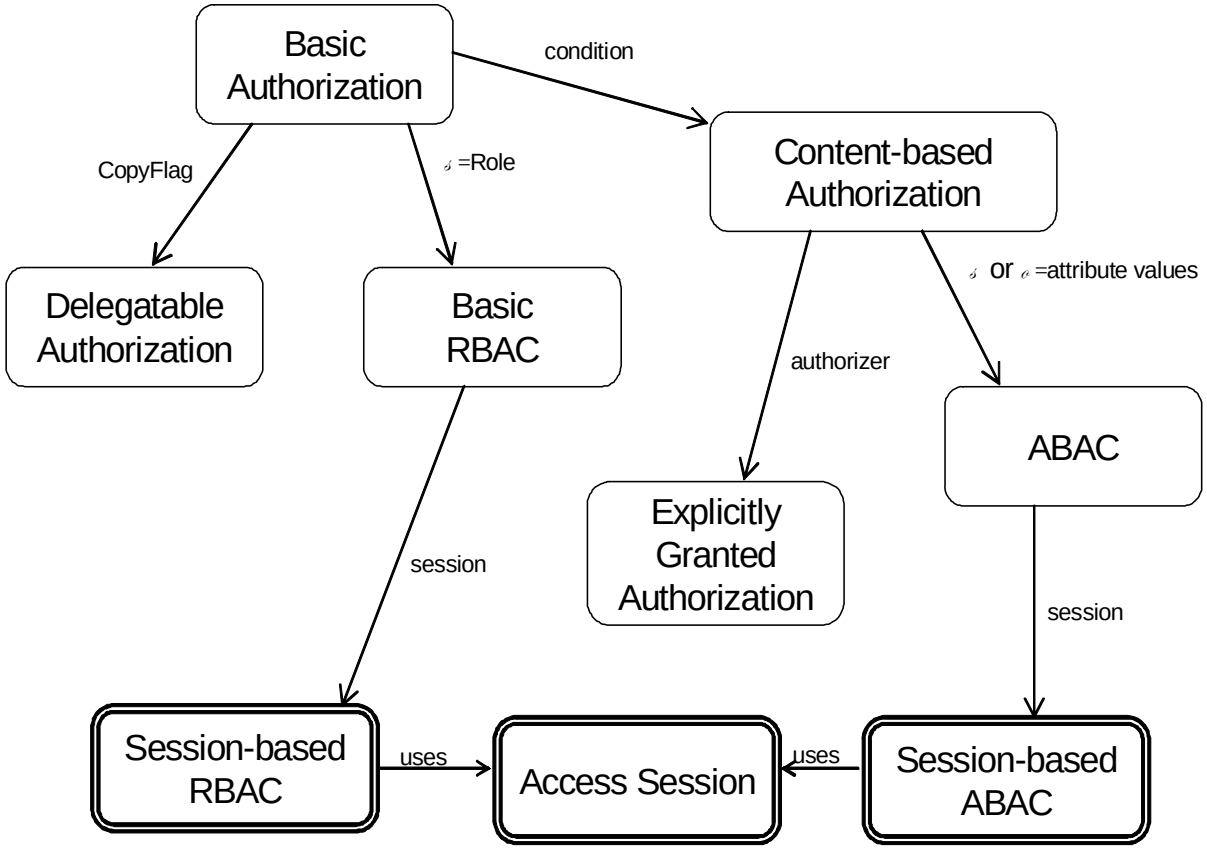
Multilevel model

- 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



Access control variations



Methodology

- Use case activities define attacks
- Attacks lead to policies to stop them
- Use cases define needed actor rights
- Access matrix or RBAC models formalize these rights
- Lower levels (defined by more patterns) enforce the rights

To stop or mitigate the attacks we need the following policies

- A1. A3. Mutual authentication. Every interaction across system nodes is authenticated.
- A2. Verify source of information.
- A4. Logging. Since the manager is using his legitimate rights we can only log his actions for auditing at a later time.
- A5. A6. Separation of administration from use of data. For example, a manager can create accounts but should have no rights to withdraw or deposit money in the account.
- A7. Protection against denial of service. We need some redundancy in the system to increase its availability. Intrusion detection and filtering policies should also be useful
- A8. Authorization. If the user is not explicitly authorized he should not be able to move money from any account.

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 operating system layers
- Example illustrates pattern templates

Anatomy of a pattern

- Patterns are described by templates
- Templates have a fixed set of sections that describe the pattern in a standard way
- We use the POSA template, there are two more: the GOF and the Alexandrian
- Remote Authenticator/Authorizer
- The LACCEI paper shows another example, a Firewall pattern

Remote Authenticator/Authorizer

- **Intent:** Provide facilities for authentication and authorization when accessing shared resources in a loosely-coupled distributed system
- **Example:** A multinational corporation may have employees, say in the US and Brazil. The user authentication and authorization information necessary to support an employee in the US is stored in the US servers and the information to support that of a Brazilian Employee is stored in Brazil servers. Now assume that an employee from the US is traveling to Brazil and has the need to access some data from the Brazilian database servers.

  There are two possible ways to achieve this

  Replicate the user information of the employee in the Brazilian Server and give her the proper authorizations to access the data.

  Borrow the username of an employee in Brazil who has similar rights and use that username to access the required information.

  Both of these solutions have their disadvantages. The system administrators will be faced with creating and managing user accounts within each of the multiple systems to be accessed in a coordinated manner in order to maintain the consistency of the security policy enforcement. If the username of another employee is borrowed, accountability is compromised

Remote authenticator II

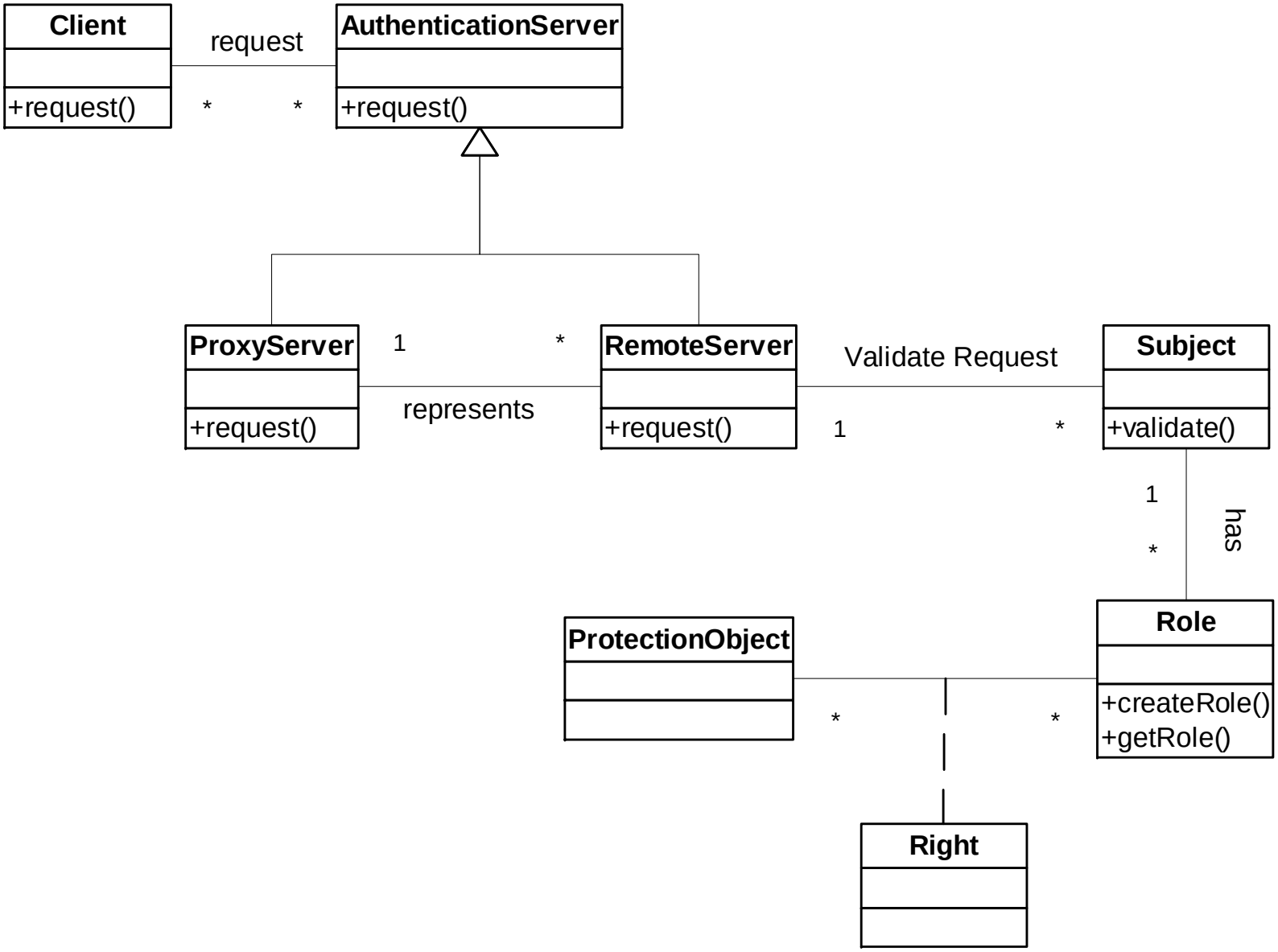
- **Context:** Loosely-coupled distributed systems such as the Internet, that consist of a variety of computational nodes, and where some nodes need to share resources. For example, a company with several divisions in different countries.
- **Problem:** How can we provide authentication and authorization in a distributed environment without the need for redundant user login information? In the past few years, telecommuting, the Internet, and electronic commerce have developed from an alternative means of doing business to become increasingly mainstream consumer activities. The concern for corporate data security has grown tremendously and the need for single user sign on to multiple domains and multiple services is becoming more of a necessity than a luxury. A system with a centralized sign-on can provide easy management, more accountability and secure authentication.

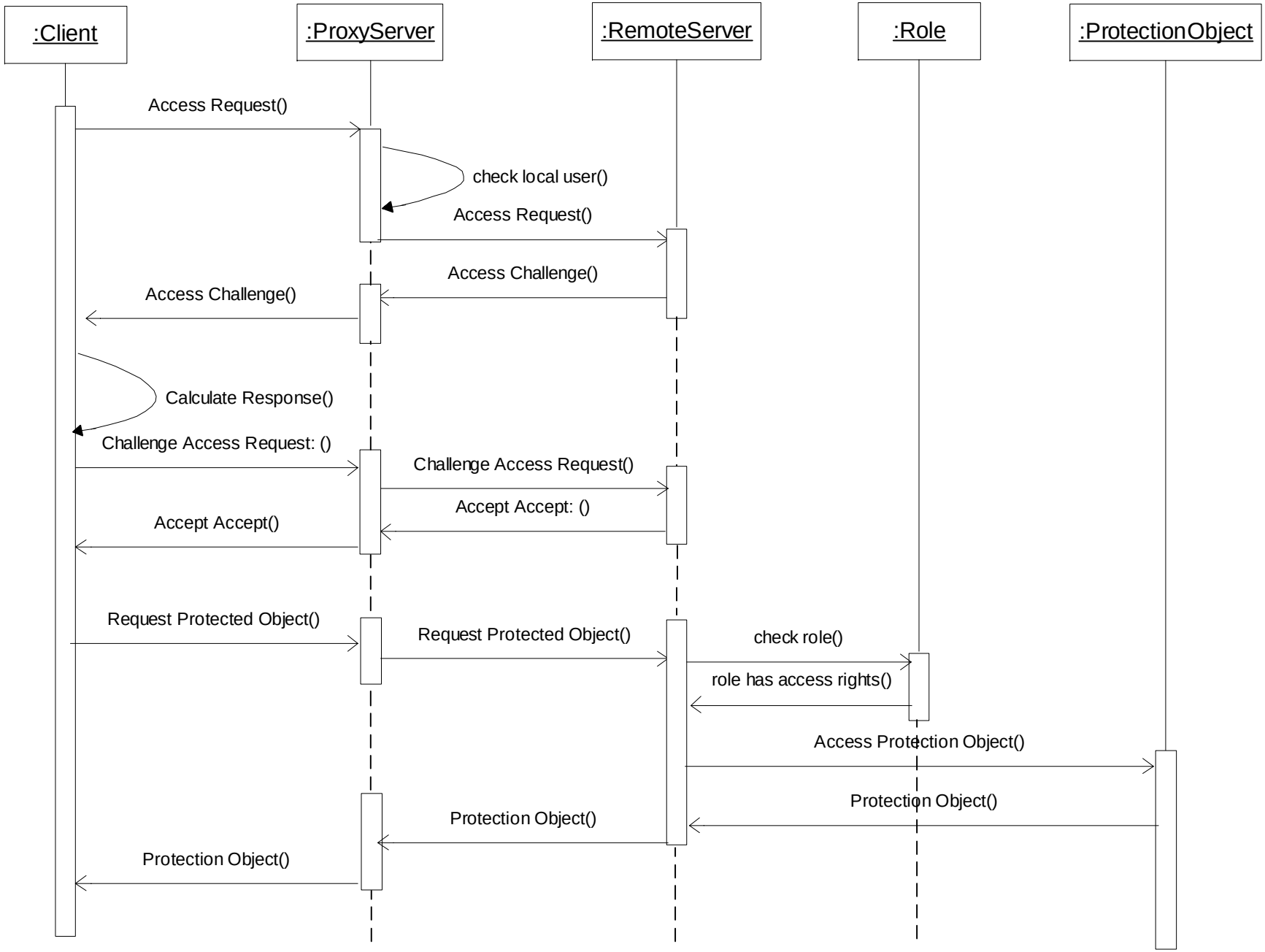
Forces

- Storing user authentication and authorization information at multiple locations makes them redundant, difficult to administer, and prone to inconsistencies.
- Although the authentication information may be stored anywhere, this location should be transparent to the users.
- Users typically work in the context of some role and these roles should be standard across a variety of domains, at least within a company or institution.
- Borrowing the login rights of a local user makes it impossible to make the user accountable, we need a way to keep the user id when he is accessing resources anywhere.

Solution

- Set up a single entry point that can transparently redirect the user to the correct server where his user login and access information can be validated.
- Use a specialized authentication/authorization server. This server is used for embedded network devices such as routers, modem servers, switches, etc. The authentication servers are responsible for receiving user connection requests, authenticating the user, and then returning all configuration information necessary for the client to deliver service to the user. The **Client** makes a request for a service through a **Proxy Server** that represents the actual server that contains the user login information. The request is routed to the **Remote Server,** which validates it, based on the **Role** of the **Subject** of the request and the **Rights** of this role with respect to the **Protection Object.**


Client
request
AuthenticationServer
+request()
*
*
+request()
ProxyServer
1
*
RemoteServer
Validate Request
Subject
+request()
represents
+request()
1
*
+validate()
1
*
has
ProtectionObject
Role
+createRole()
+getRole()
*
*
Right


:Client
:ProxyServer
:RemoteServer
:Role
:ProtectionObject
Access Request()
check local user()
Access Request()
Access Challenge()
Access Challenge()
Calculate Response()
Challenge Access Request: ()
Challenge Access Request()
Accept Accept: ()
Accept Accept()
Request Protected Object()
Request Protected Object()
check role()
role has access rights()
Access Protection Object()
Protection Object()
Protection Object()
Protection Object()

Consequences

This pattern has the following advantages:

- Roaming permits two or more administrative entities to allow each other's users to dial in to either entity's network for service.
- Storing the user login and access rights at a single location makes it more secure and easy to maintain.
- The user's login ID, password etc. are stored in the internal radius database or can be accessed from an SQL Database.
- The location where the user information is stored is transparent to the user.
- Roles and access rights have to be standard across locations.
- Both servers and clients should support the base protocol.
- Units such as active cards [ACS] allow complex request/challenge interactions.

There are also some liabilities:

- The additional messages used increase overhead, thus reducing performance for simple requests.
- The system is more complex than a system that directly validates clients.

Remote Authenticator

- **Implementation:** An authentication server can function as both a forwarding server and a remote server, serving as a forwarding server for some realms and a remote server for other realms. One forwarding server can act as a forwarder for any number of remote servers. A remote server can have any number of servers forwarding to it and can provide authentication for any number of realms. One forwarding server can forward to another forwarding server to create a chain of proxies. A lookup service is necessary to find the remote server.
- **Example resolved**: When the US employee travels to Brazil he logs in a Remote Authenticator/Authorizer which reroutes her requests to the US server that stores her login information.

Known Uses

Remote Authentication Dial-In User Service (RADIUS) is a widely deployed IETF protocol enabling centralized authentication, authorization, and accounting for network access [Has02, Rig00]. Originally developed for dial-up remote access, RADIUS is now supported by virtual private network (VPN) servers, wireless access points, authenticating Ethernet switches, Digital Subscriber Line (DSL) access, and other network access types [Hil]. Figure 3 shows the typical authentication sequence of a client in a RADIUS server using a challenge response approach..

With proxy RADIUS, one RADIUS server receives an authentication (or accounting) request from a RADIUS client (such as a NAS), forwards the request to a remote RADIUS server, receives the reply from the remote server, and sends that reply to the client. A common use for proxy RADIUS is roaming. Roaming permits two or more administrative entities to allow each other's users to dial in to either entity's network for 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

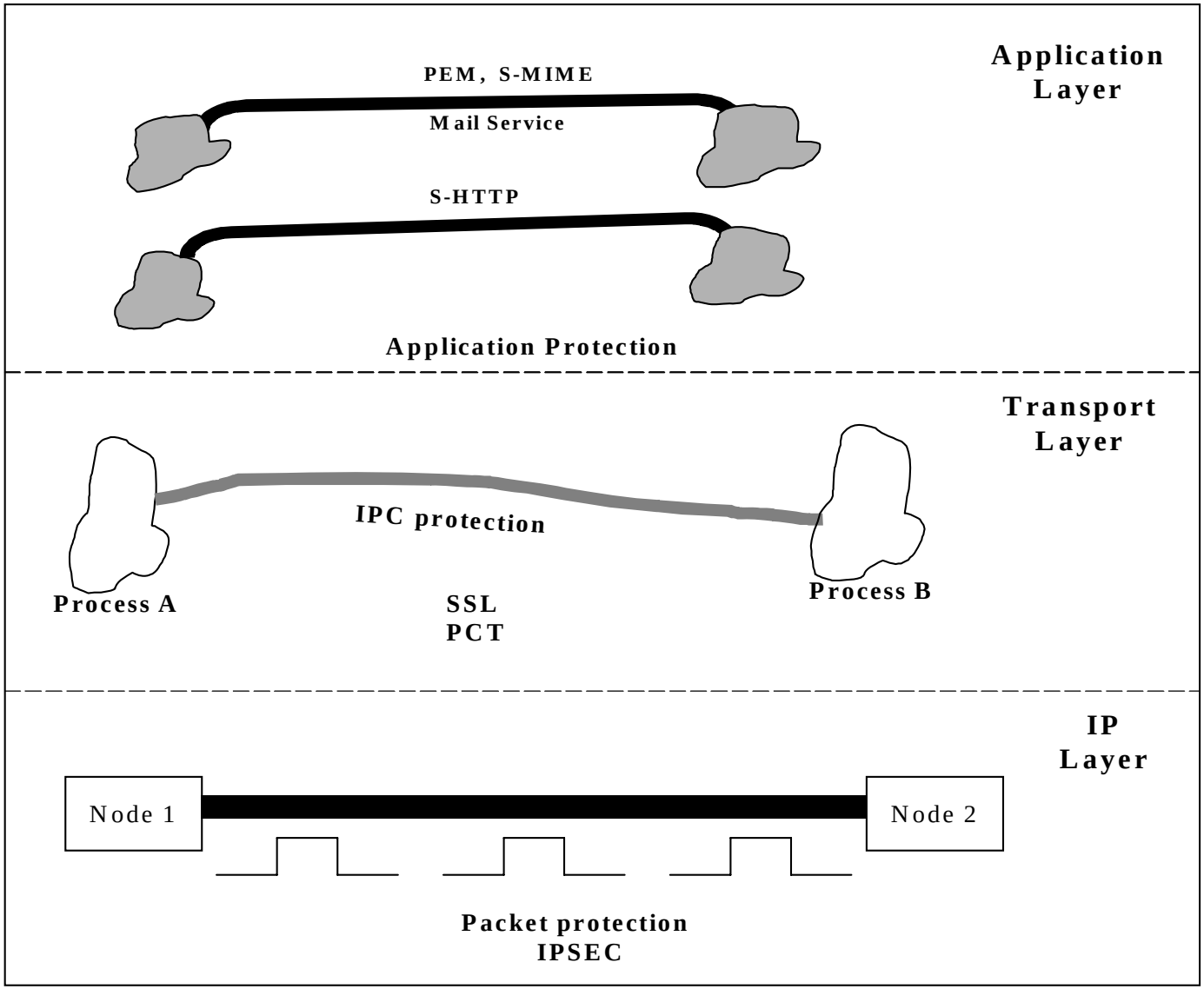
The network

- Contact with the outside world
- Send and receive messages, files, web pages,…
- Unknown users
- Communication mechanisms are part of the operating system: ports, sockets,…
- Layered architecture

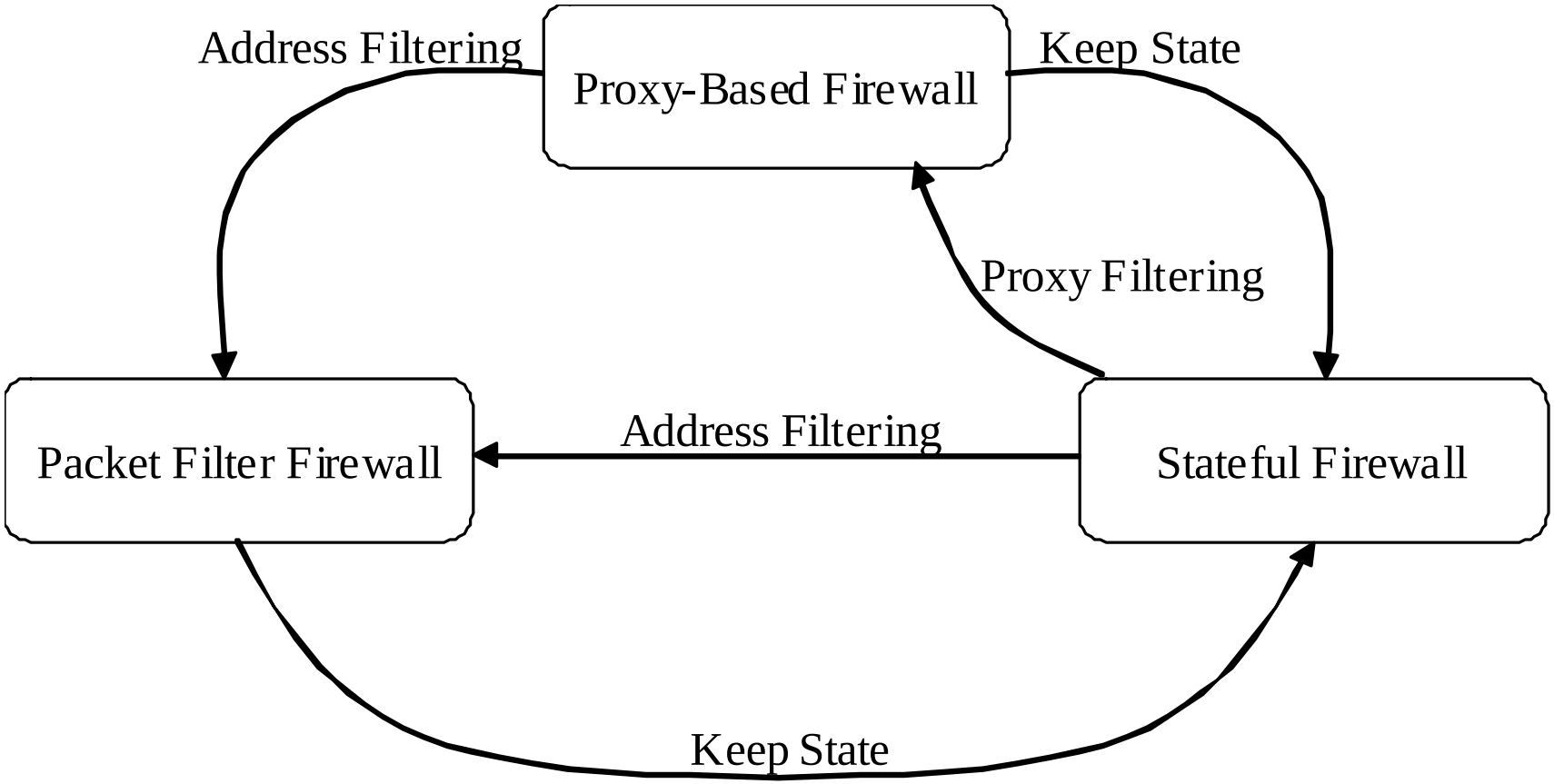
Internet layers

- Layer 7 (HTTP),
- Layer 4 (TCP, Transmission Control Protocol),
- Layer 3 (IP, Internet Protocol),
- Layer 1.
- At the higher levels, the sub-protocols used are TCP (a connection-oriented protocol), and UDP (User Datagram Protocol)

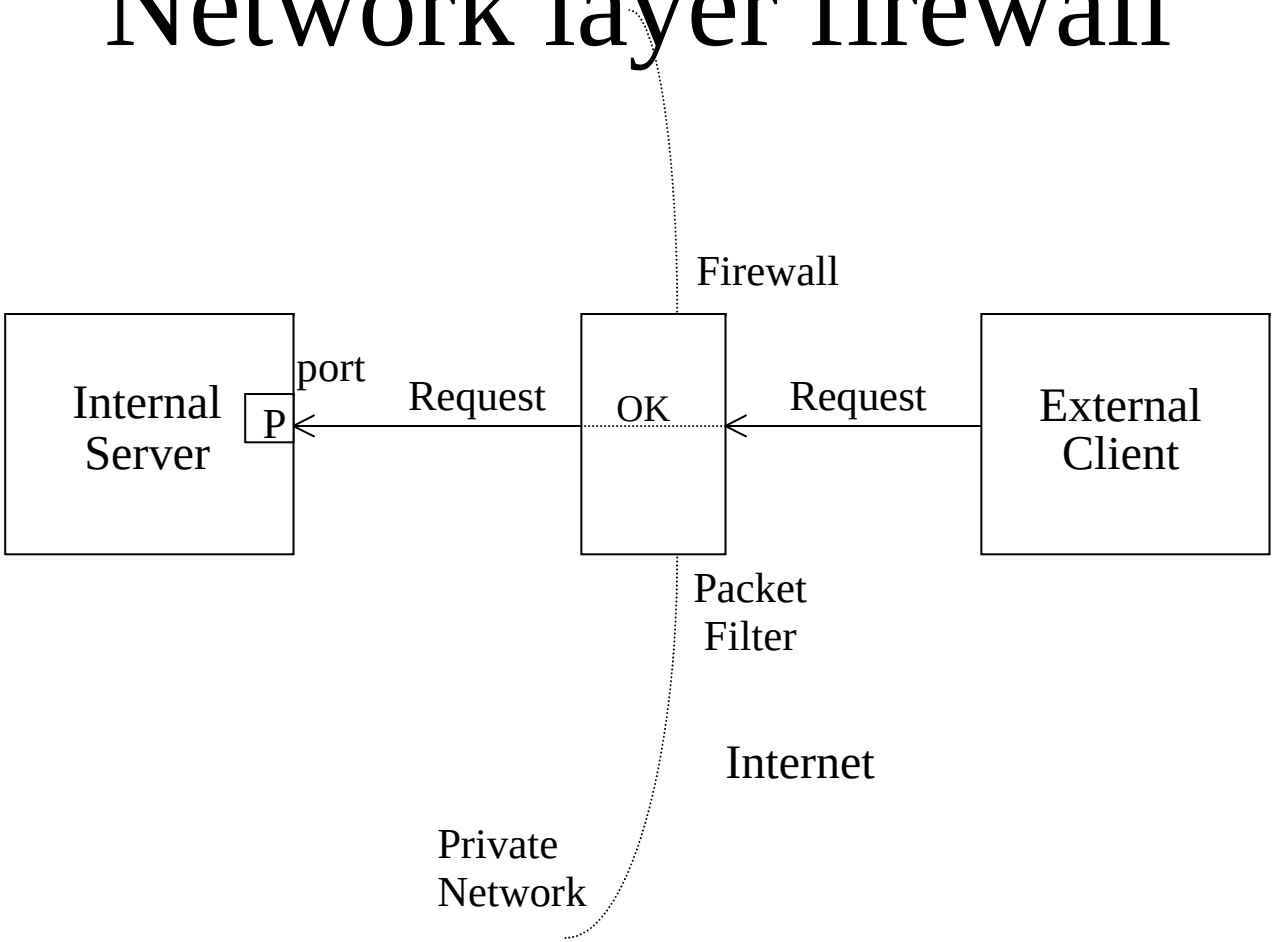
Secure channels

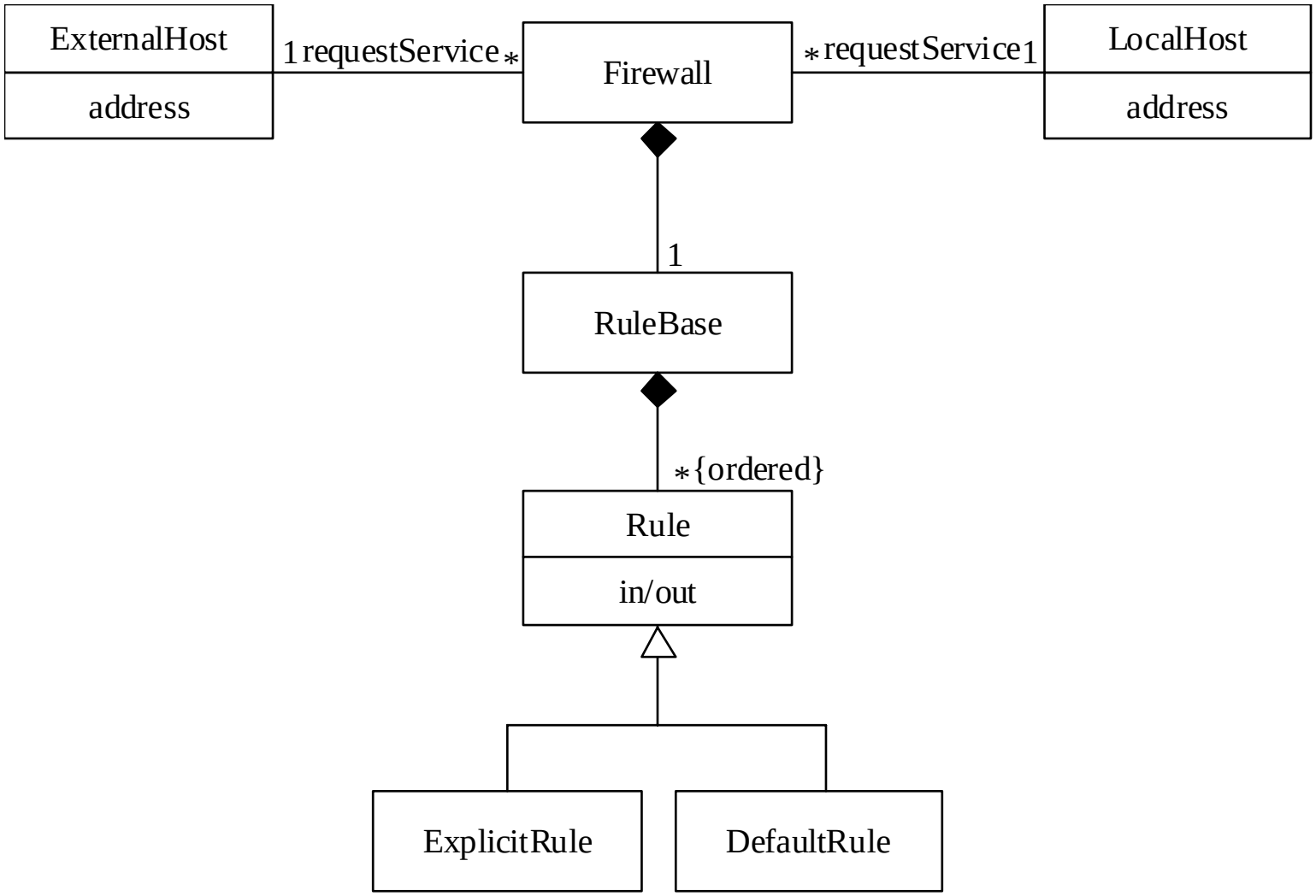


Firewall patterns

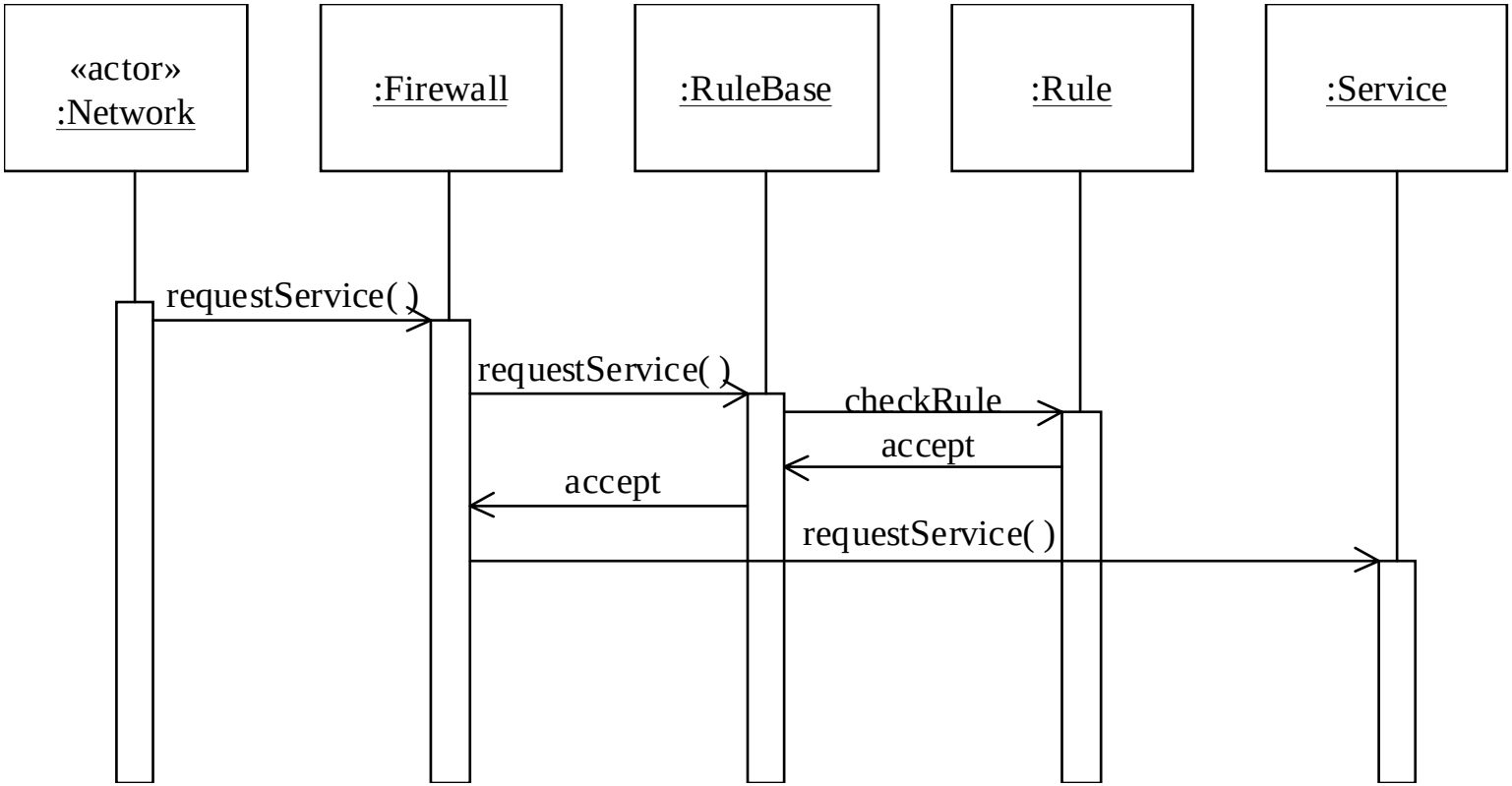


Network layer firewall




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

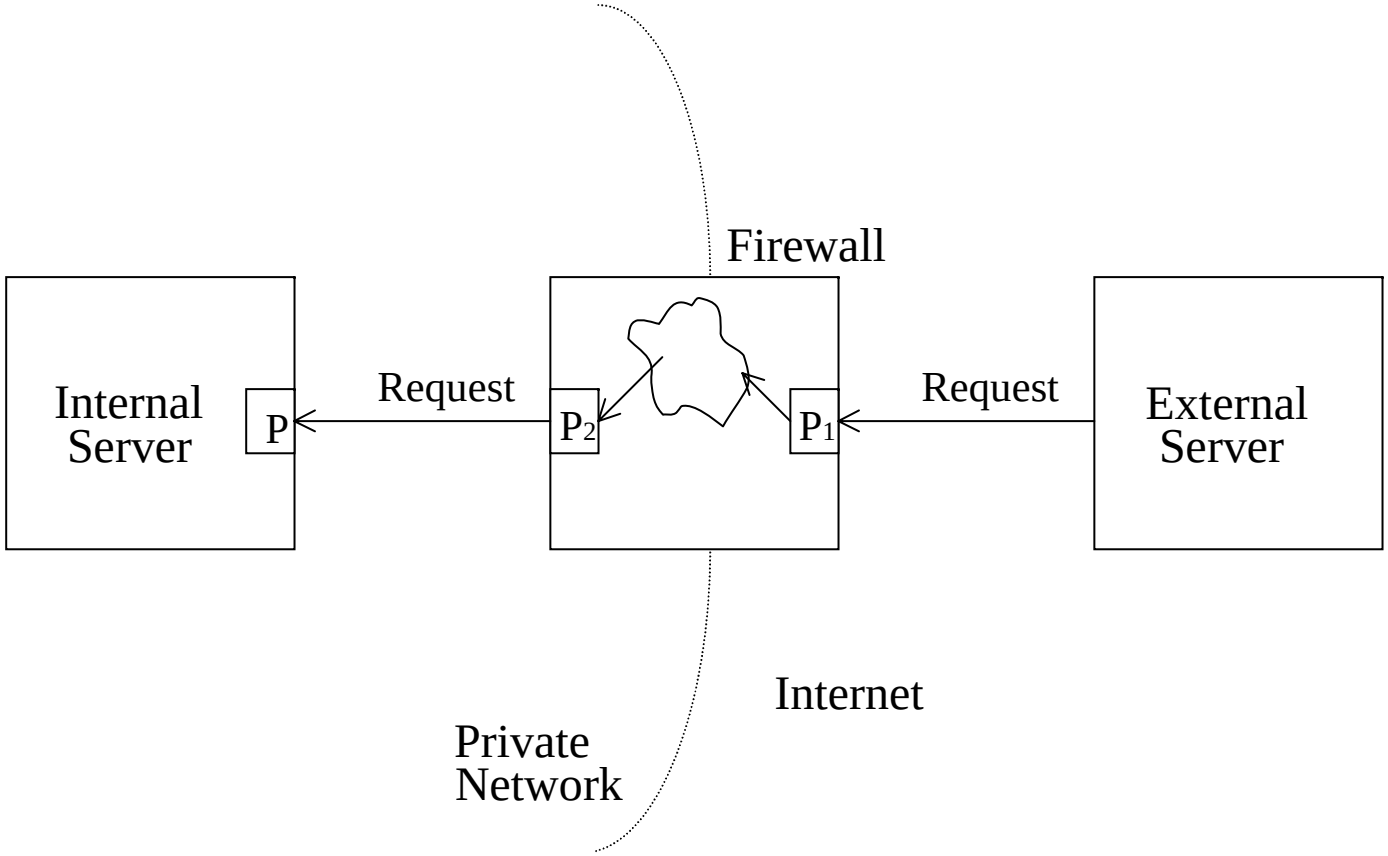
Filtering a request



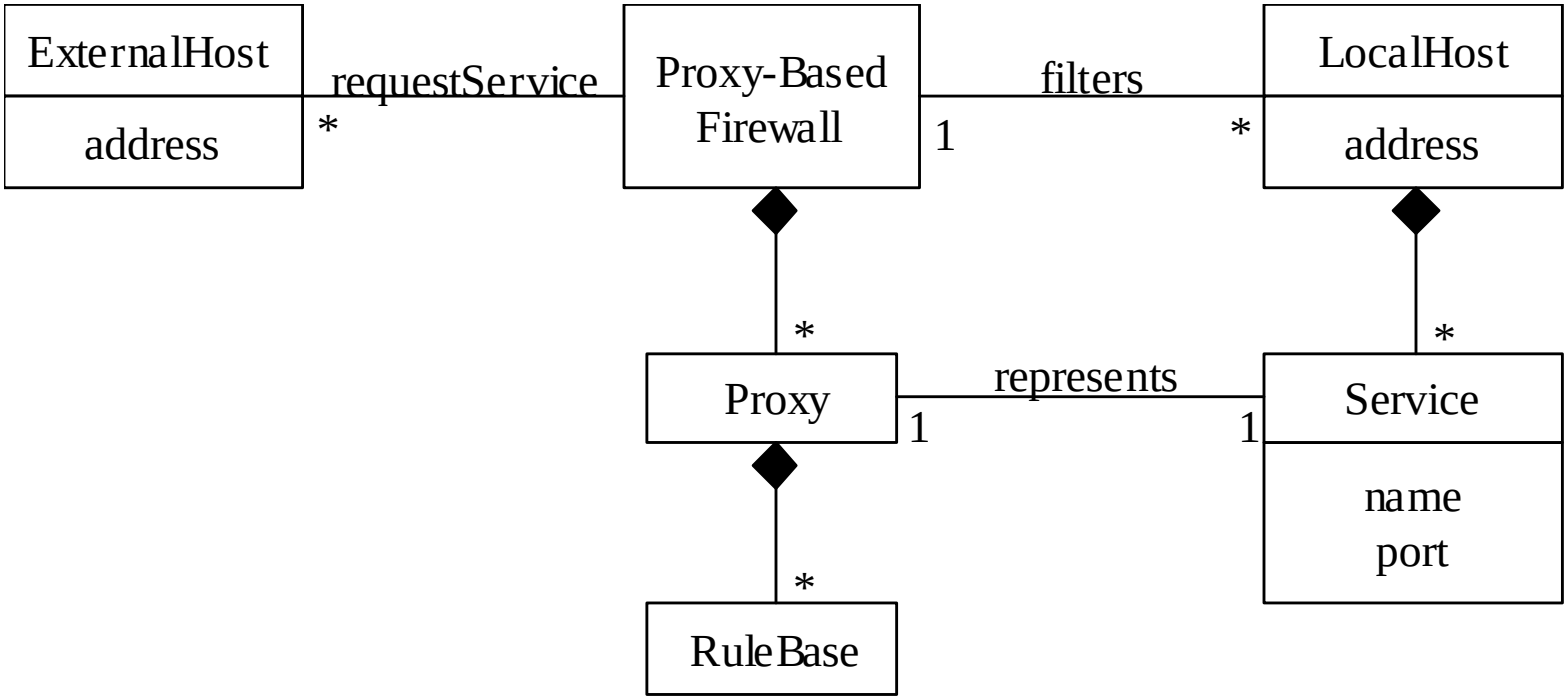
Application layer (proxy) firewall

- 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
- Can be complemented with intrusion detection

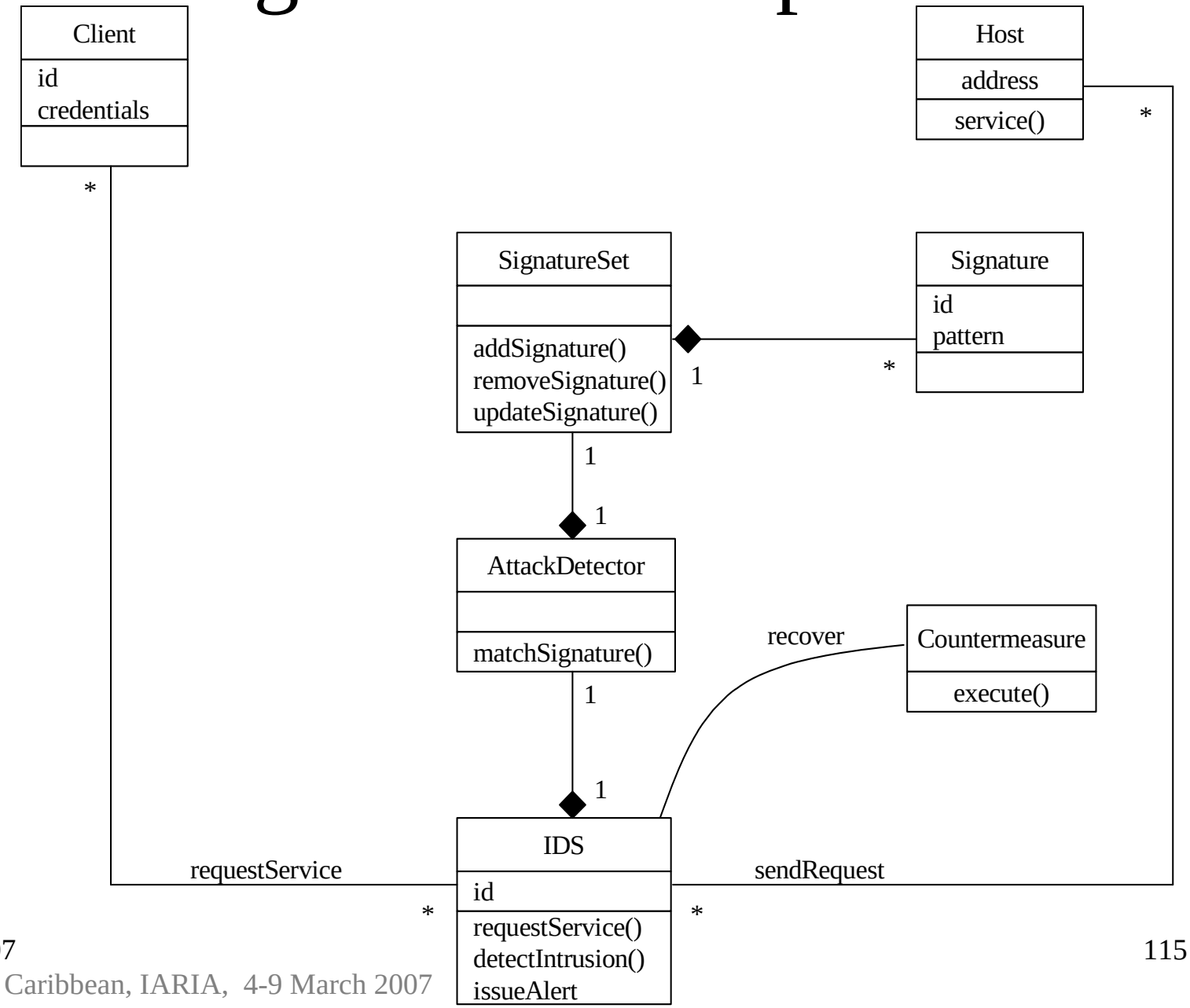
Intrusion Detection Systems (IDS)

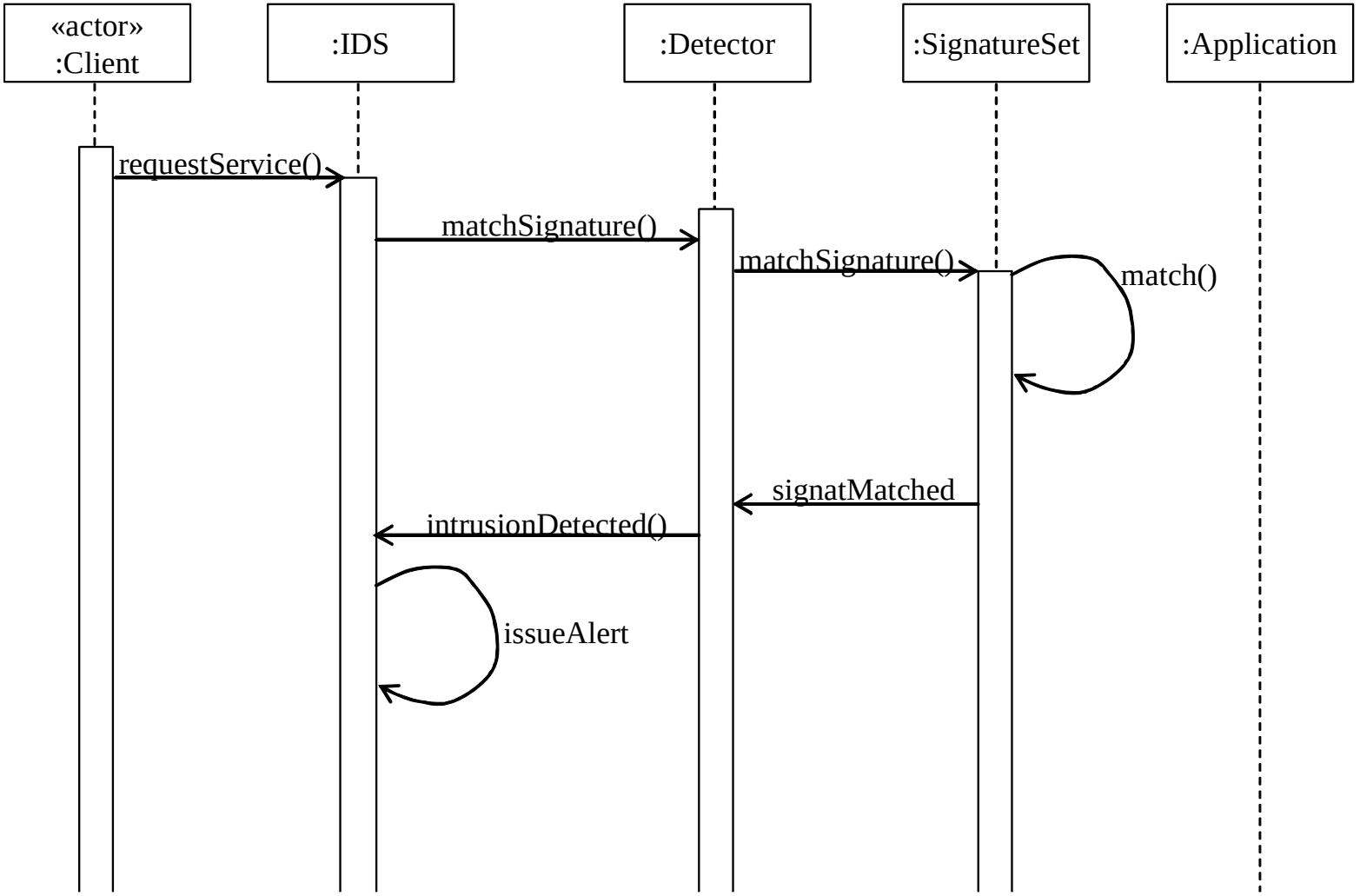
- Try to detect an ongoing attack
- React to attack
- Can use knowledge about past attacks (signature)
- Can use deviations of “normal” behavior
- Can be host-based or network-based

How do they work?

- *Statistical anomaly detection* --based on profiles of normal user and system behavior. Events that deviate from this behavior are considered suspicious. The profiles are built from past audit logs.
- *Rules-based detection* (knowledge based)-- based on sequences of events (attack signatures), that correspond to known types of attack.

Knowledge-based IDS pattern




«actor»
:Client
:IDS
:Detector
:SignatureSet
:Application
requestService()
matchSignature()
matchSignature()
match()
signatMatched
intrusionDetected()
issueAlert

Virtual Private Networks

- Based on cryptographic tunneling -- from client to server directly or through tunnel-enabled access servers
- Tunneling protocols : Microsoft PPTP and Cisco L2F
- Some products do authentication of tunnel end points
- At level 4 (SSL) or 2(IP)

Layer defenses

XML	XML Firewall	XML VPN
Application	Application Firewall	—
Layer 7	Proxy-Based Firewall	SSL VPN
Layer 3	Packet-Filter Firewall	IP VPN

New patterns

- All types of VPNs
- IPSEC
- SSL (TLS)

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

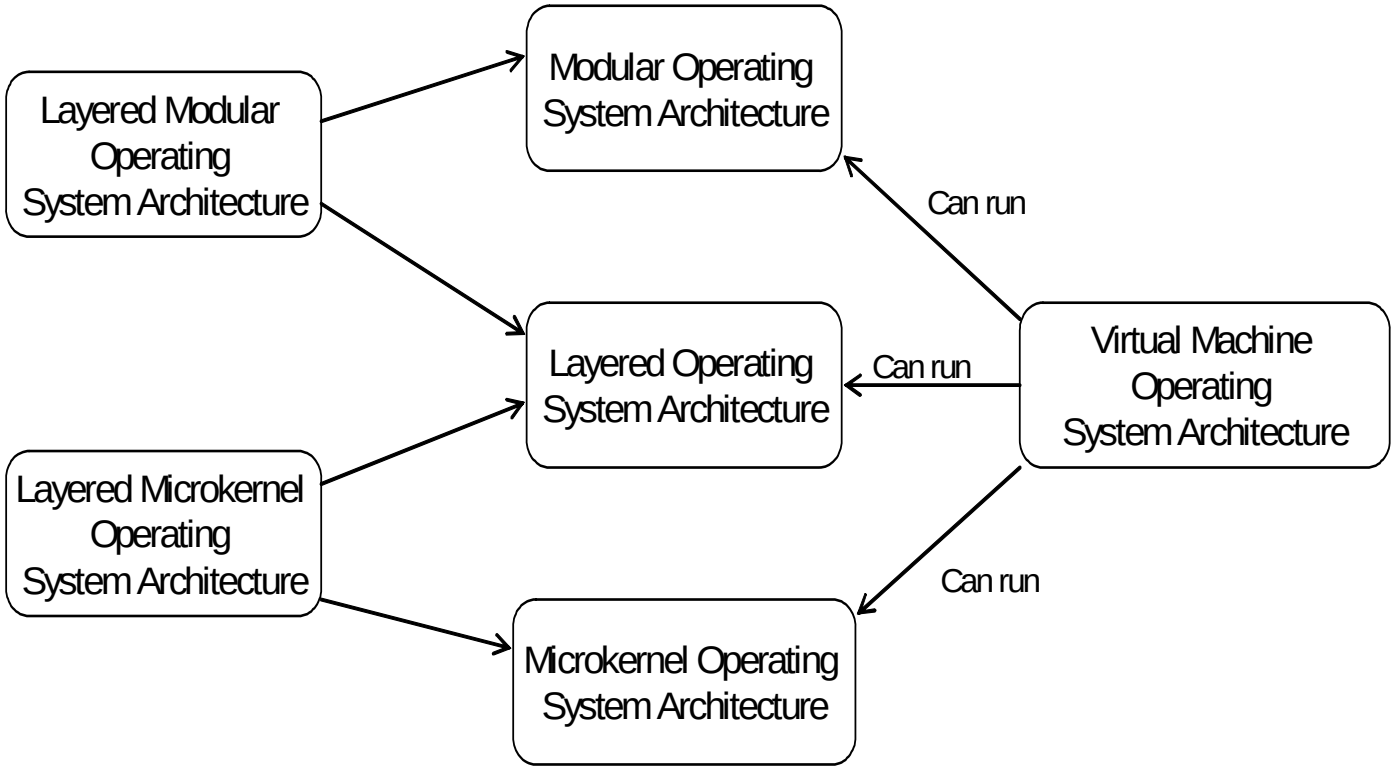
- Remote login weaknesses
- Password guessing
- Bypass file permissions
- Scavenge memory
- Buffer overflow attacks
- Denial of service attacks (resource hogging)
- Privileged CGI scripts (in HTTP server OS)

OS defenses

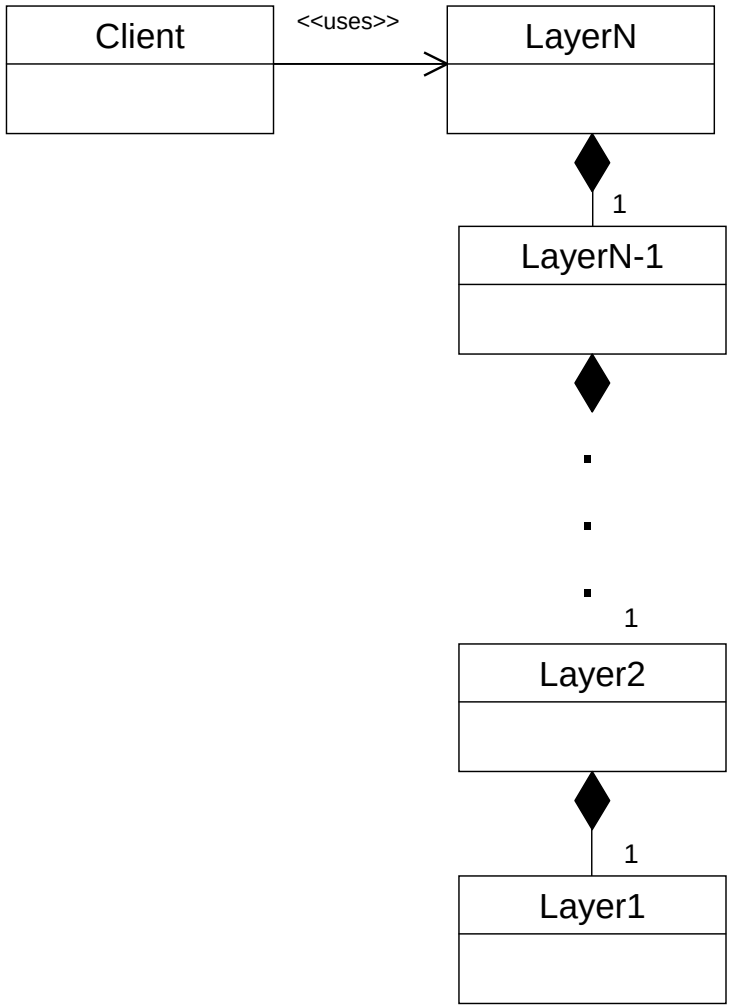
- Memory protection (supported by hardware)
- File protection
- Access control for I/O devices
- Requires good processor support for low overhead and to avoid bypassing of high-level mechanisms
- Capabilities and descriptors are effective mechanisms
- Firewalls to protect access to the system
- Authentication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

Patterns for OS secure architectures

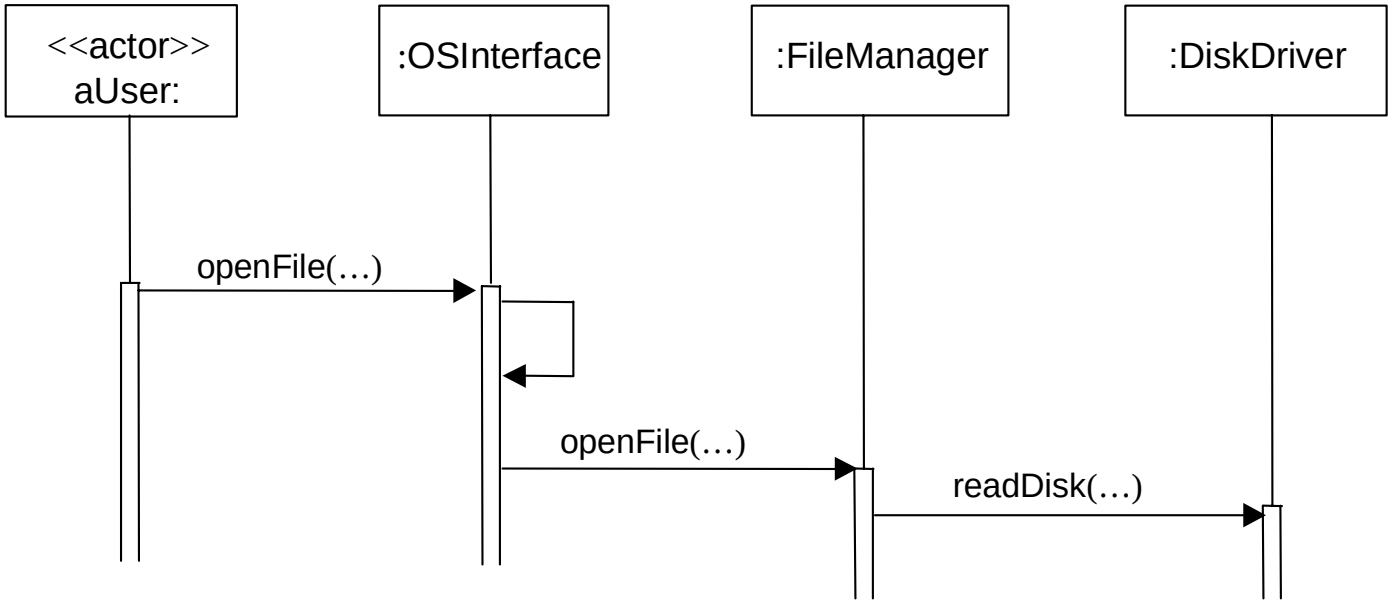
- Modular OS
- Layered OS
- Microkernel
- Virtual machine OS


Layered Modular Operating System Architecture
Modular Operating System Architecture
Can run
Layered Operating System Architecture
Can run
Virtual Machine Operating System Architecture
Layered Microkernel Operating System Architecture
Can run
Microkernel Operating System Architecture

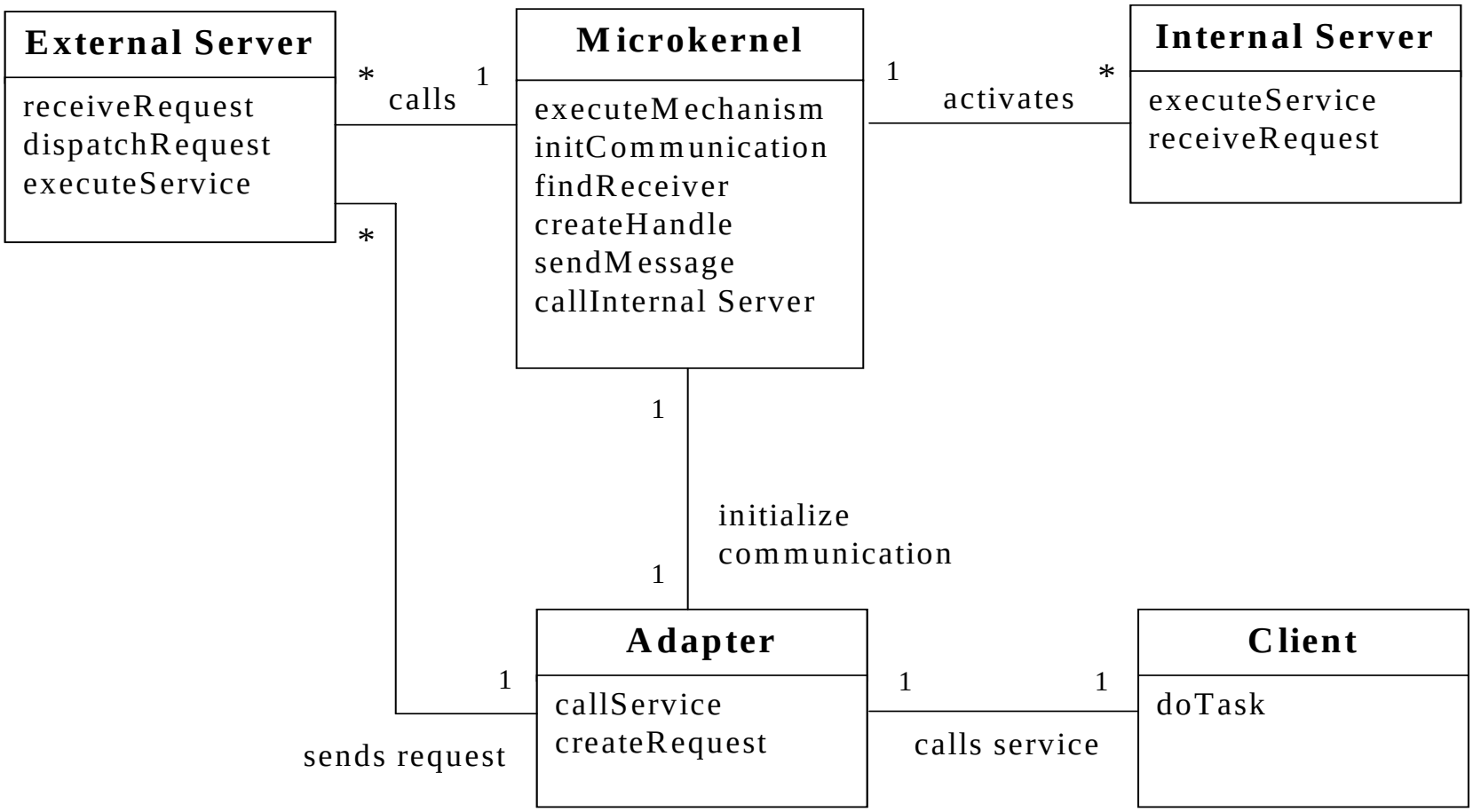
Layered OS



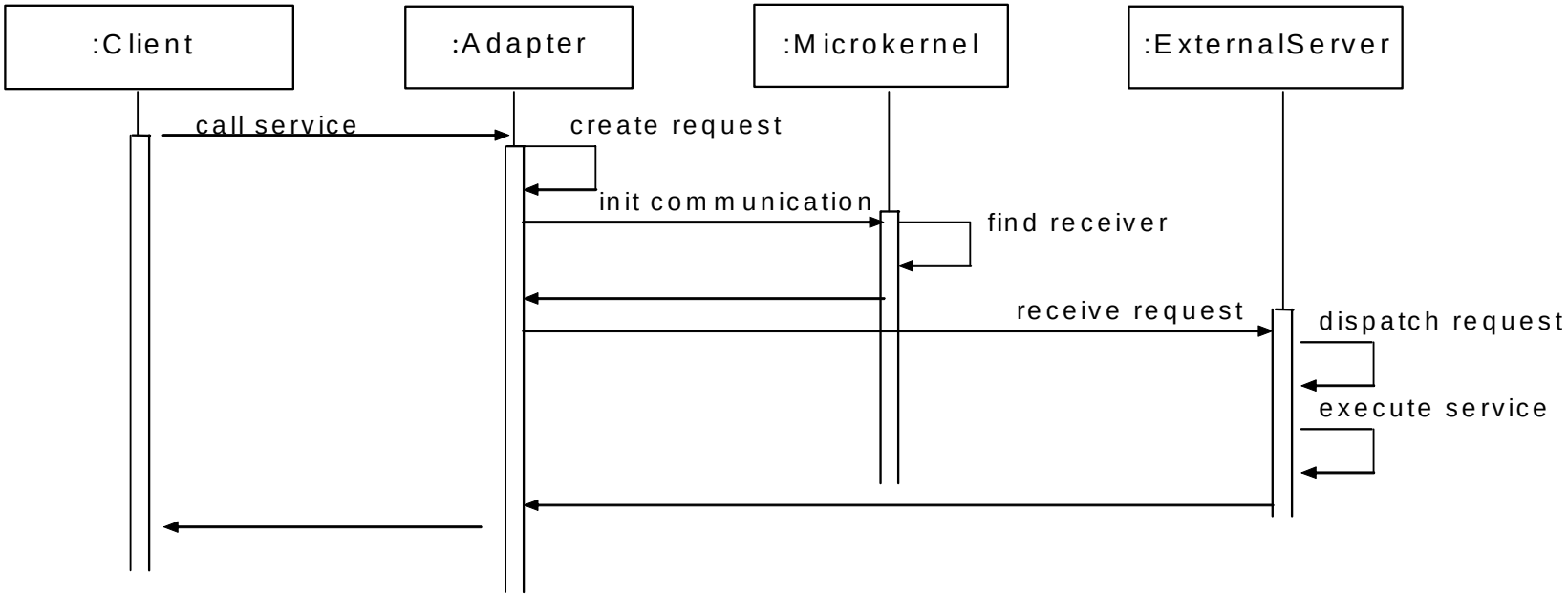
Requesting a service



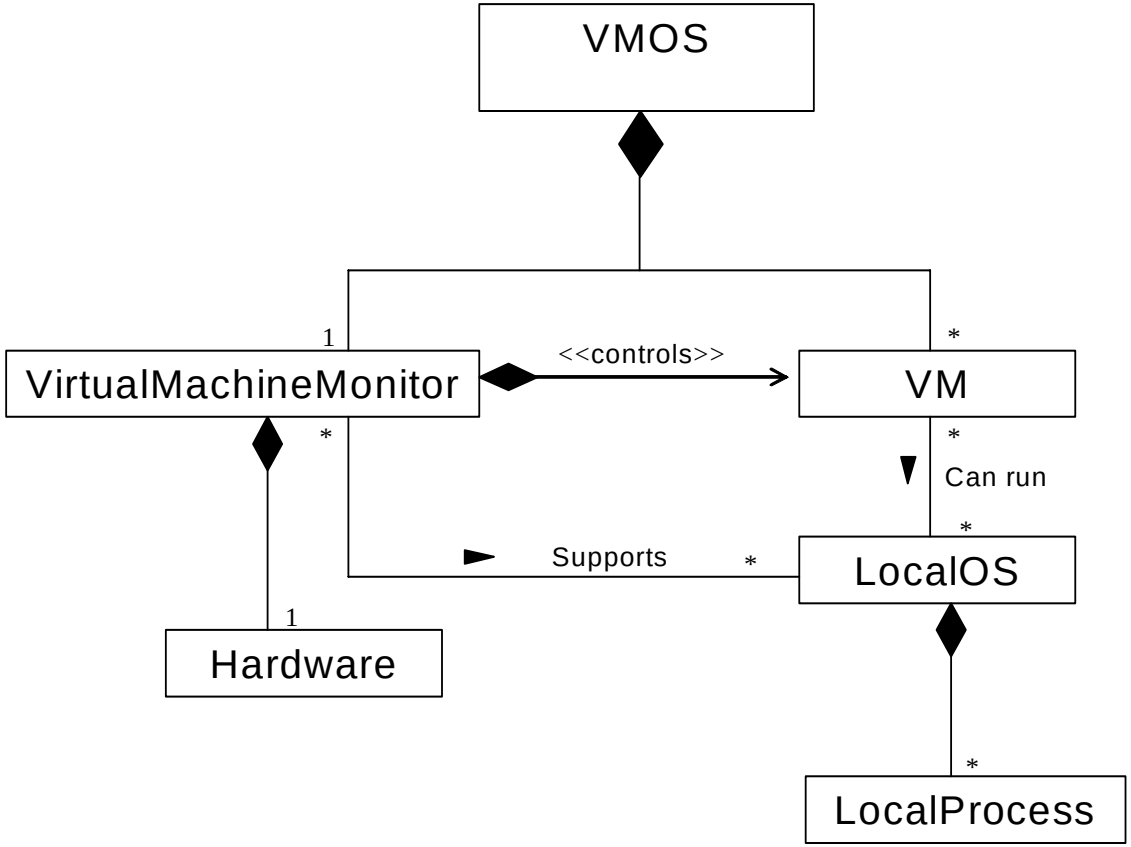
Microkernel



Requesting a service

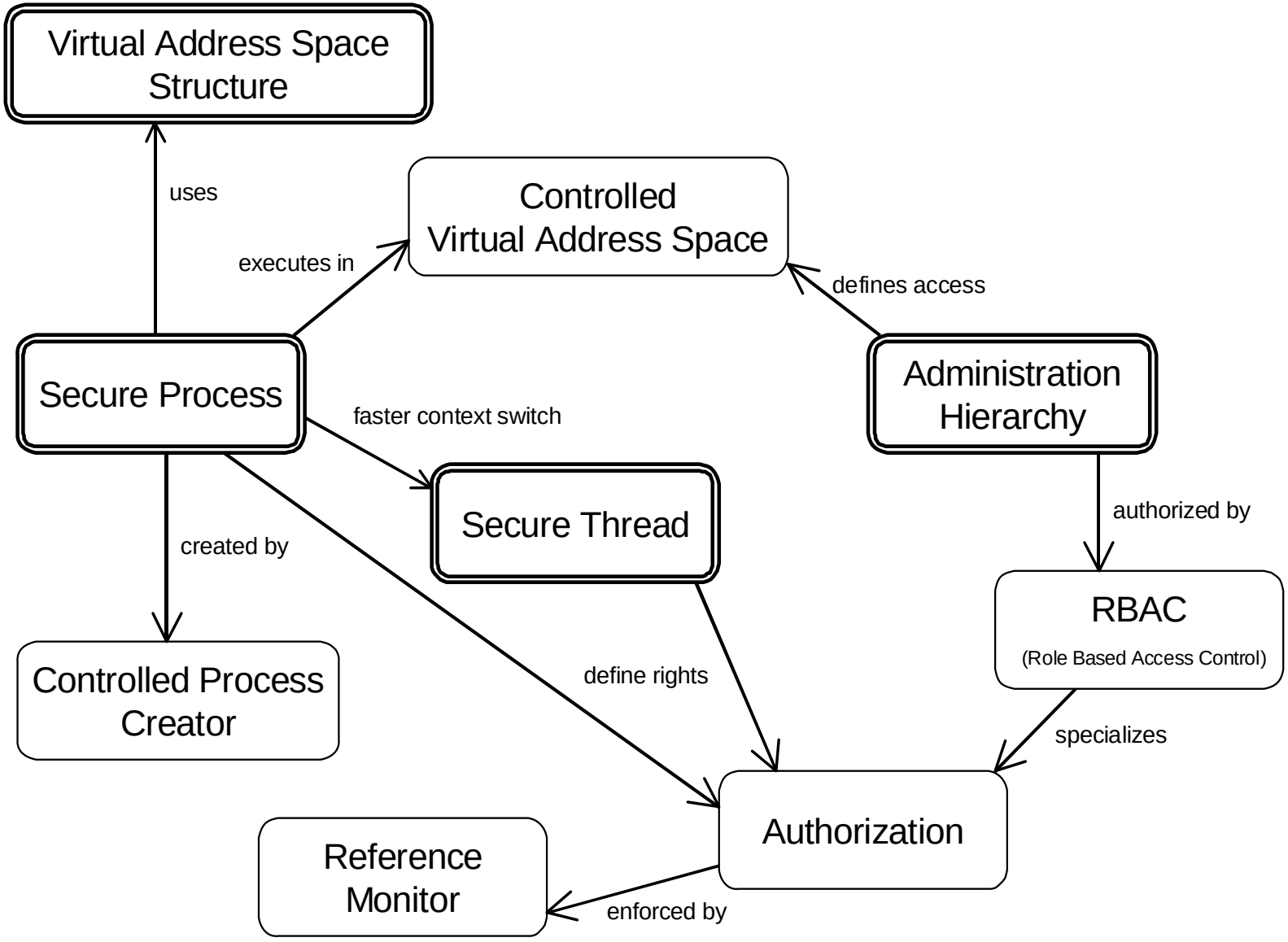


VM OS

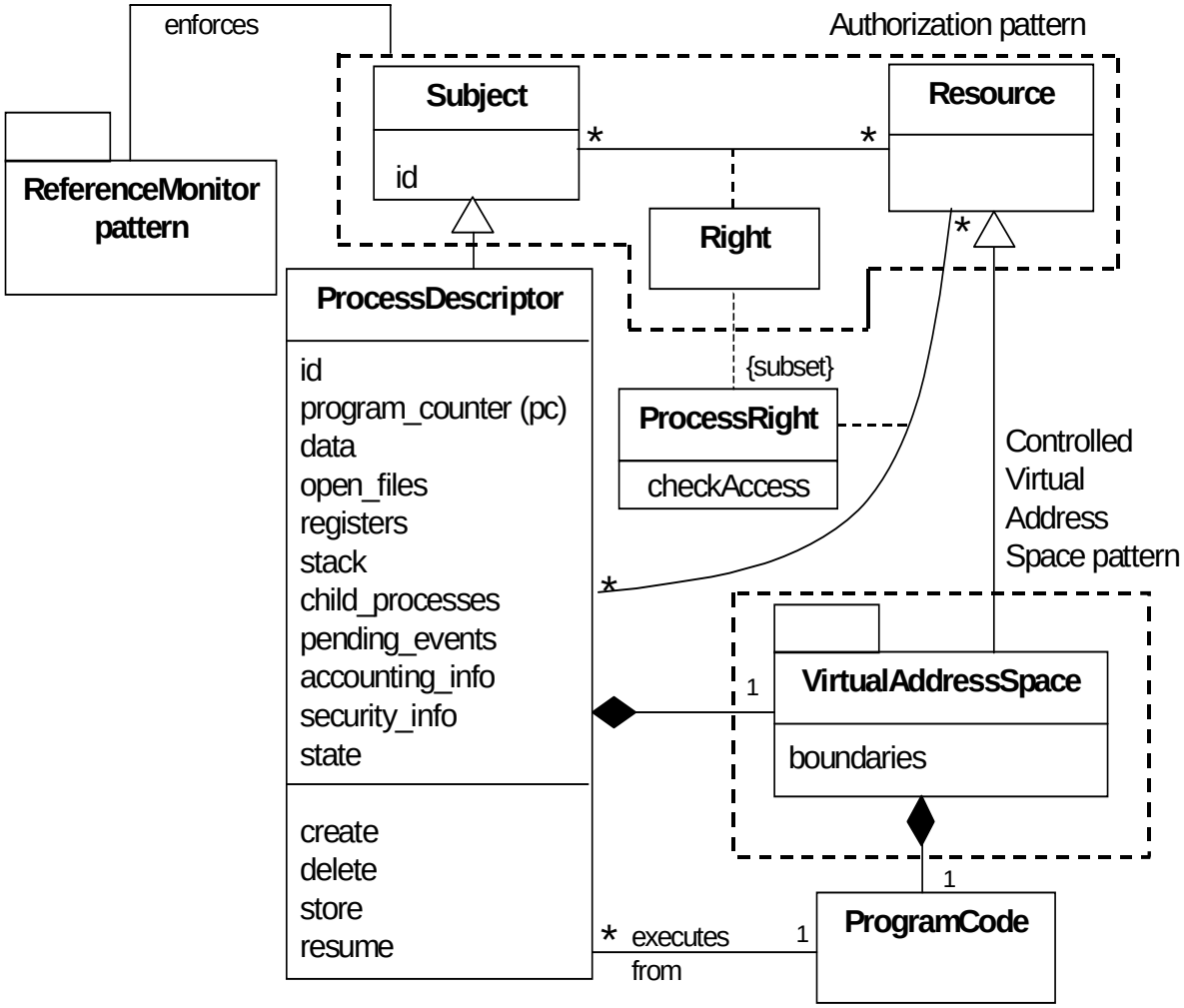

VMOS
1
*
VirtualMachineMonitor
<<controls>>
VM
*
*
Can run
*
Supports
*
LocalOS
1
Hardware
*
LocalProcess

Patterns for operating systems security

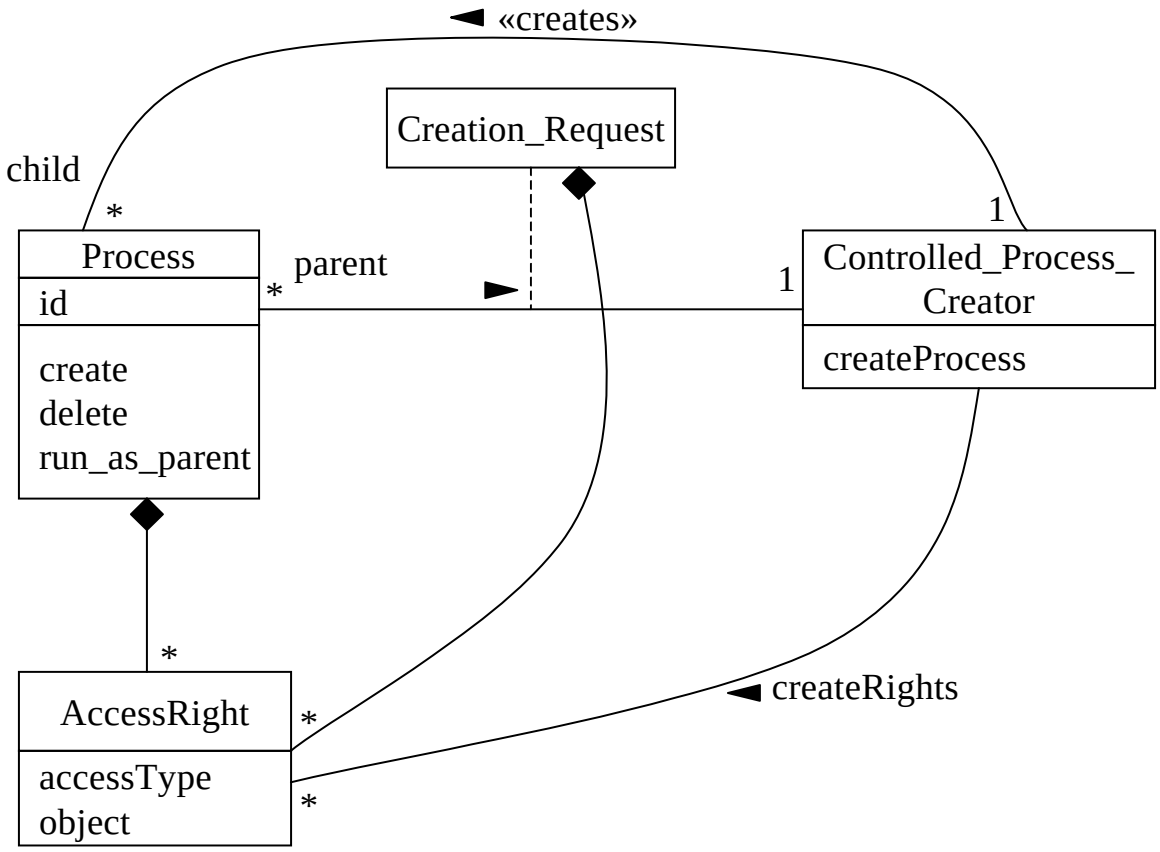
- Controlled process creation
- Controlled object creation
- Authentication
- Controlled object access (reference monitor)
- File access control
- Controlled execution environment


Virtual Address Space Structure
uses
Controlled Virtual Address Space
executes in
defines access
Secure Process
Administration Hierarchy
faster context switch
Secure Thread
authorized by
created by
RBAC
(Role Based Access Control)
Controlled Process Creator
define rights
specializes
Authorization
Reference Monitor
enforced by

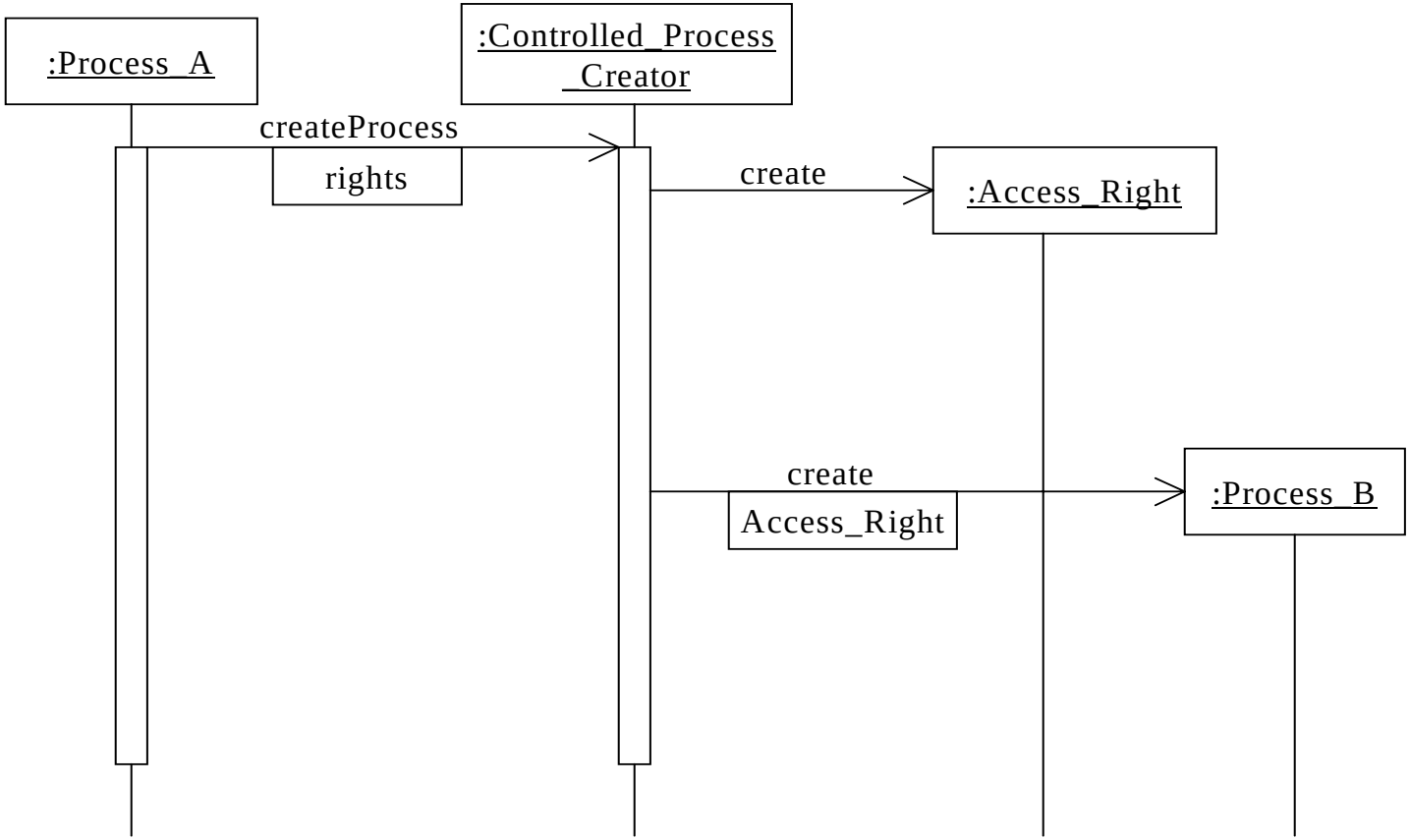
Secure Process



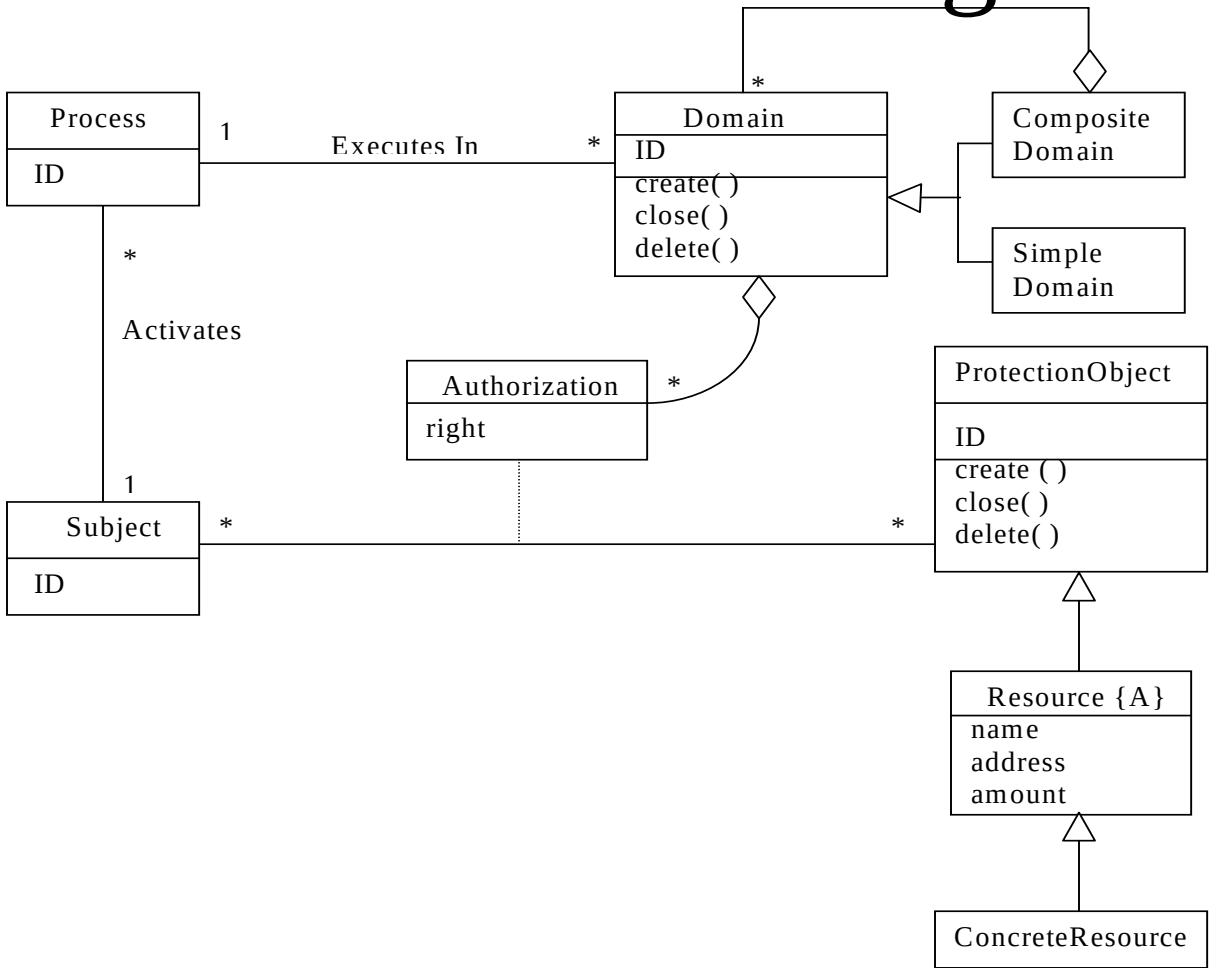
Controlled-Process Creator



Process creation dynamics



Process/domain rights



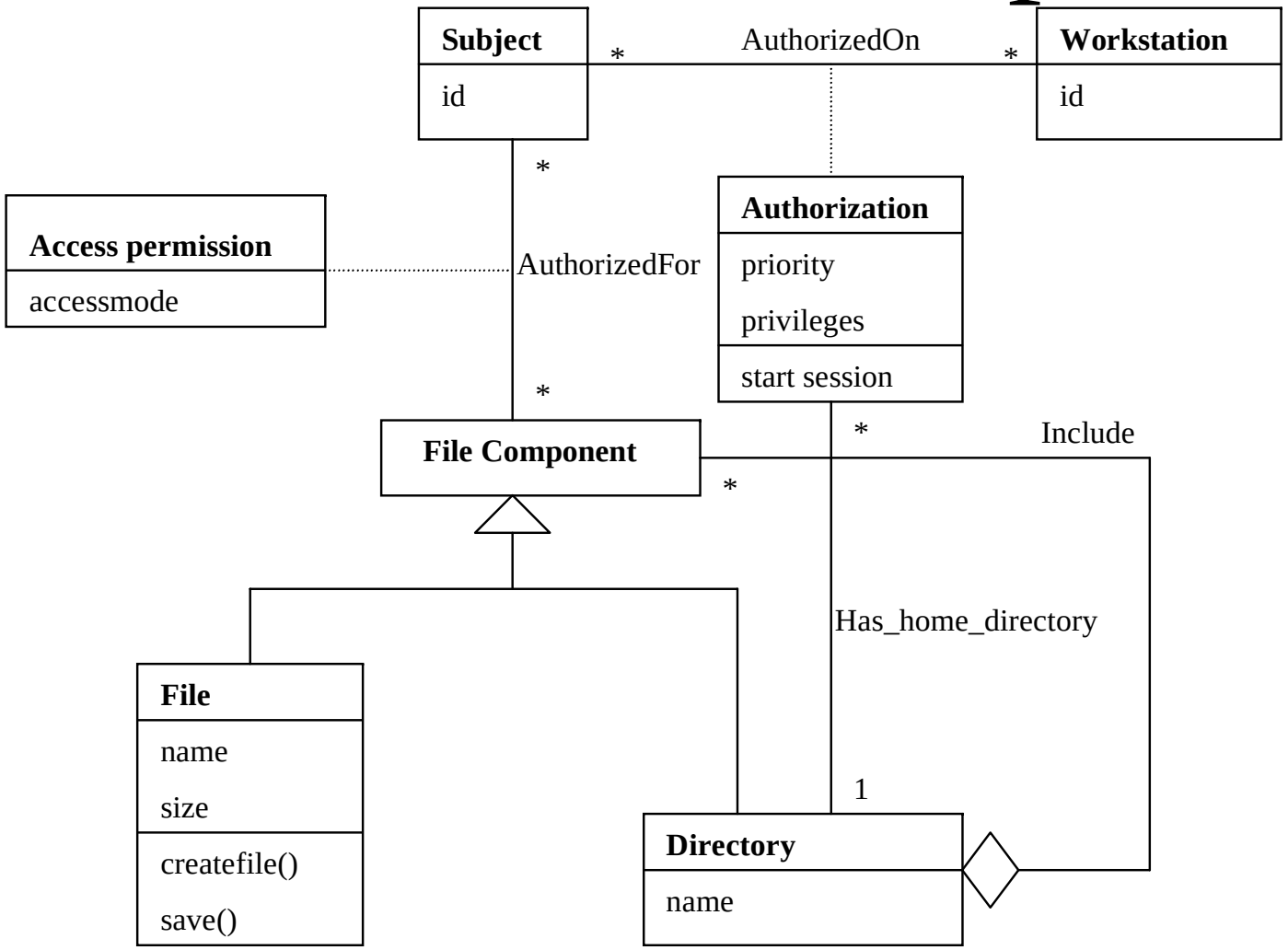
Process rights

- Access Control Lists (ACLs)—defined with each resource
- Capabilities ---defined for each process and kept by the process, enforced through hardware
- Patterns under development

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



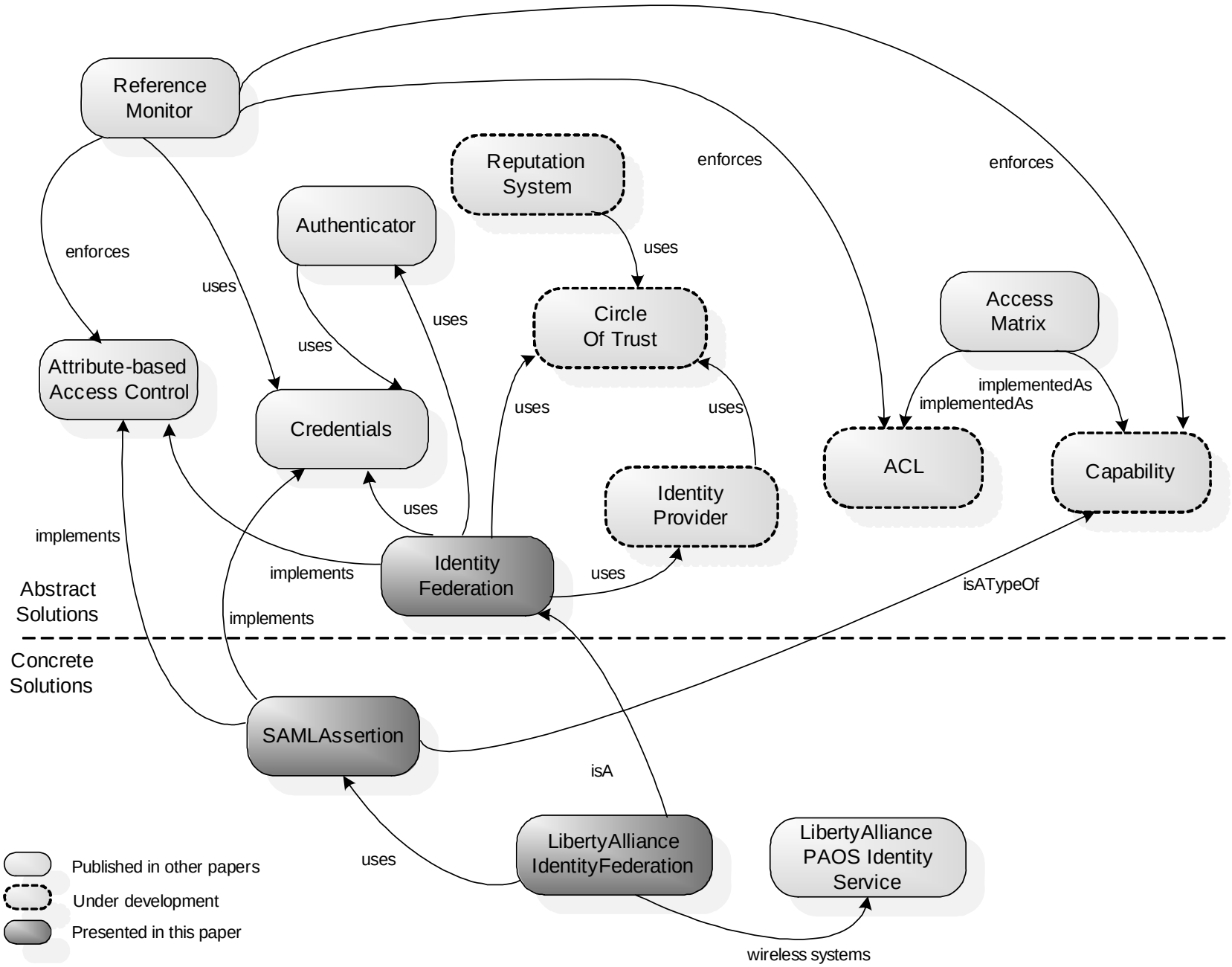
Use of subpatterns

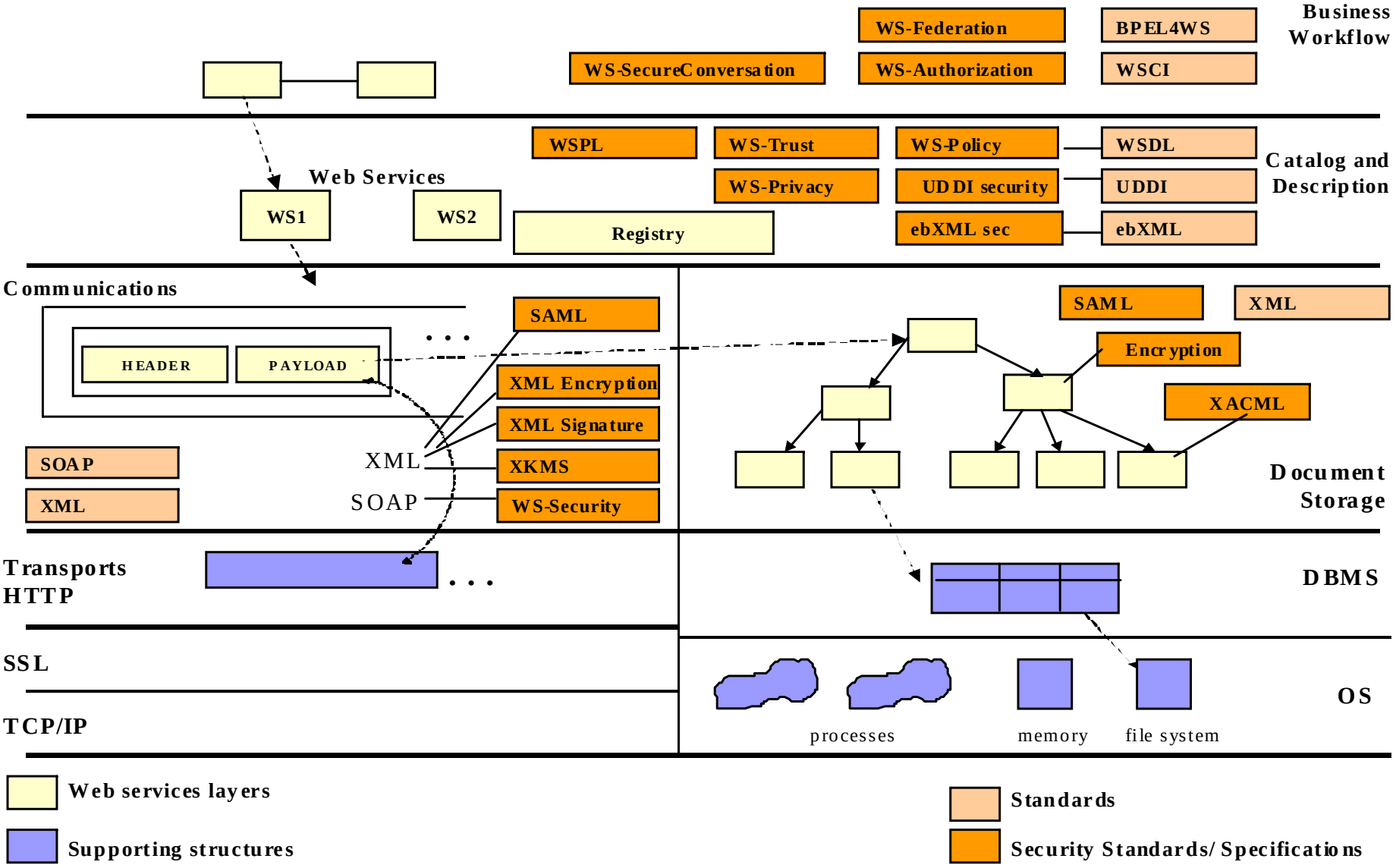
- This pattern uses two instances of the Authorization Rule pattern
- Also uses the Composite pattern (GOF)
- A higher-level authorization rule that uses objects included in specific files can be mapped to this level for enforcement

Patterns for web services

- SAML
- XACML
- XML Firewall

Patterns can be used to compare or understand standards

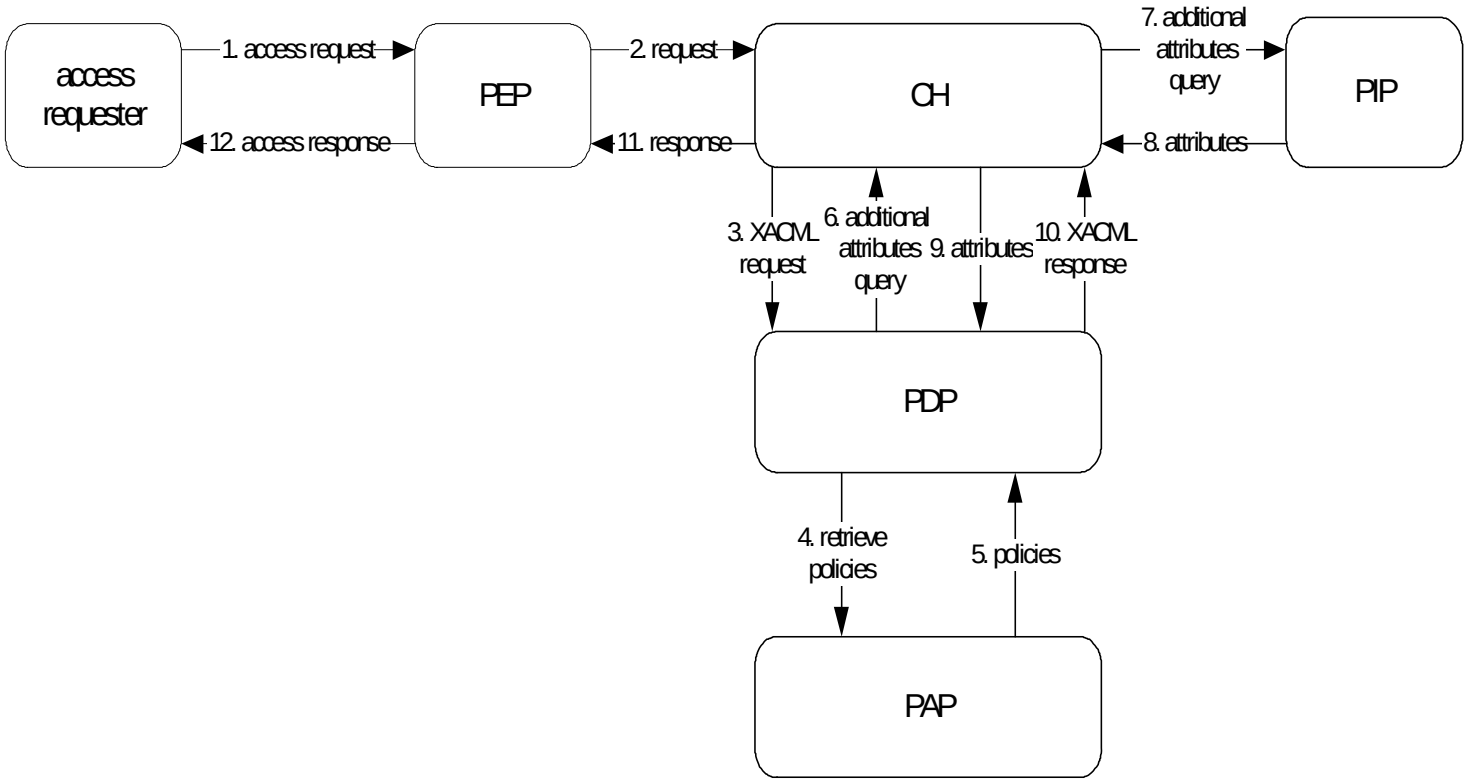

Reference Monitor
Reputation System
Authenticator
Circle Of Trust
Access Matrix
Attribute-based Access Control
Credentials
ACL
Capability
Identity Provider
Identity Federation
SAMLAssertion
LibertyAlliance IdentityFederation
LibertyAlliance PAOS Identity Service
enforces
uses
implementedAs
implements
isA
isATypeOf
Abstract Solutions
Concrete Solutions
wireless systems
Published in other papers
Under development
Presented in this paper


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
OS
TCP/IP
processes
memory
file system
Web services layers
Standards
Supporting structures
Security Standards/ Specifications

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

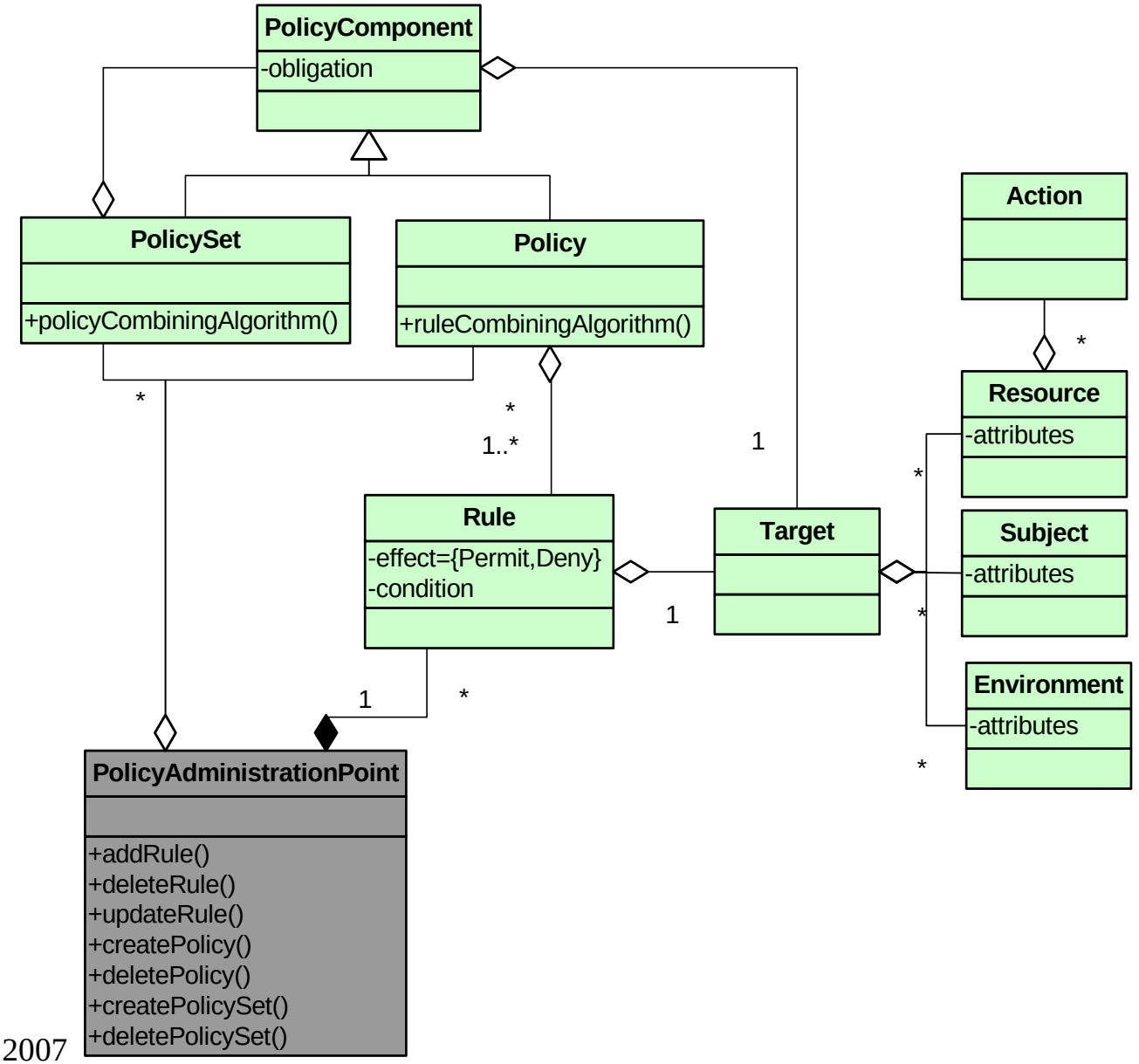
Security model



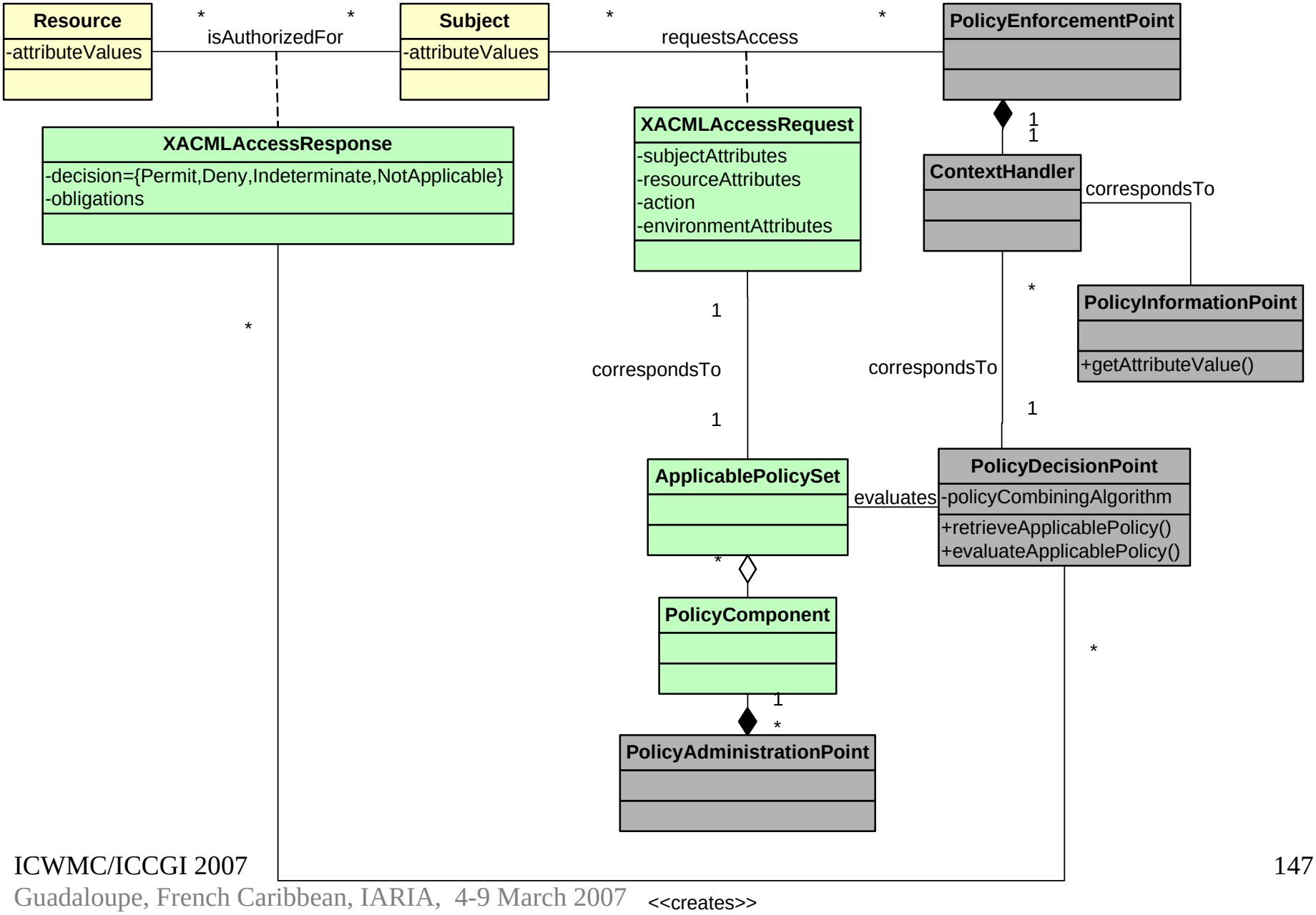
A structure for enforcement

- PEP= Policy Enforcement Point, where access control is enforced
- CH= Context handler, defines context or domain
- PDP= Policy Definition Point
- PAP= Policy Authorization Point, set of policies to authorize request
- PIP= Policy Information Point, additional information

XACML Authorization

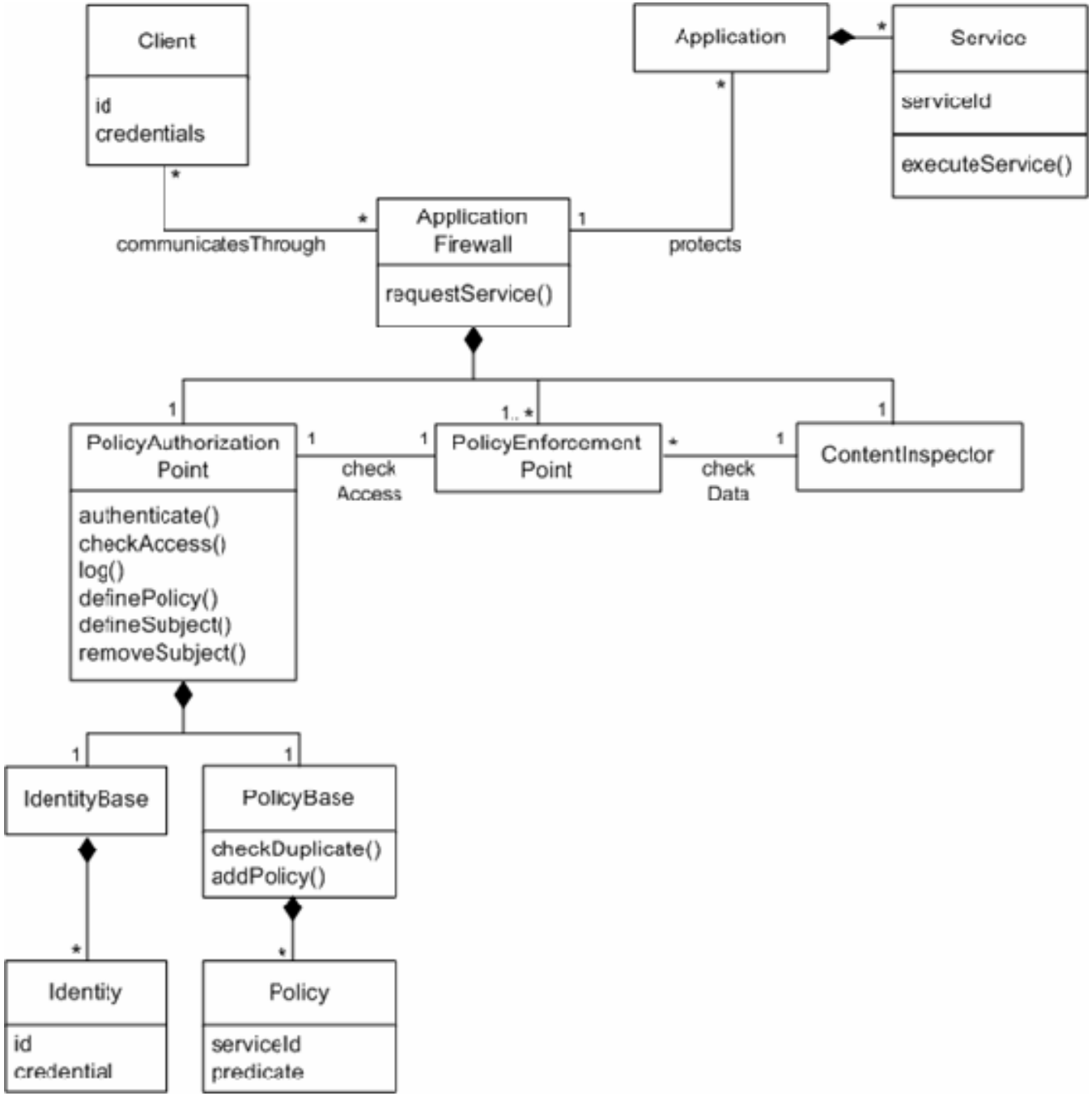


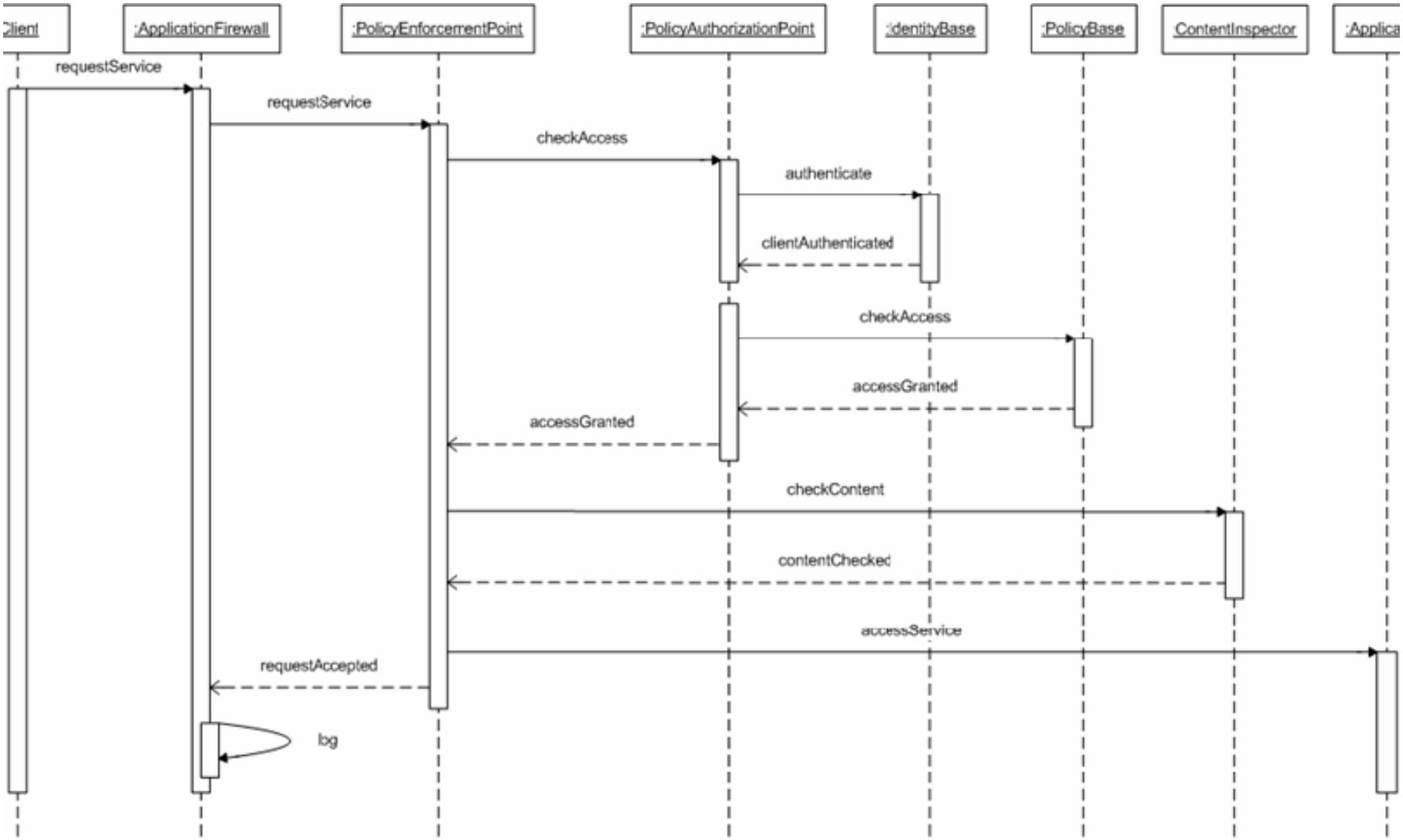
Access control evaluation



Application firewall

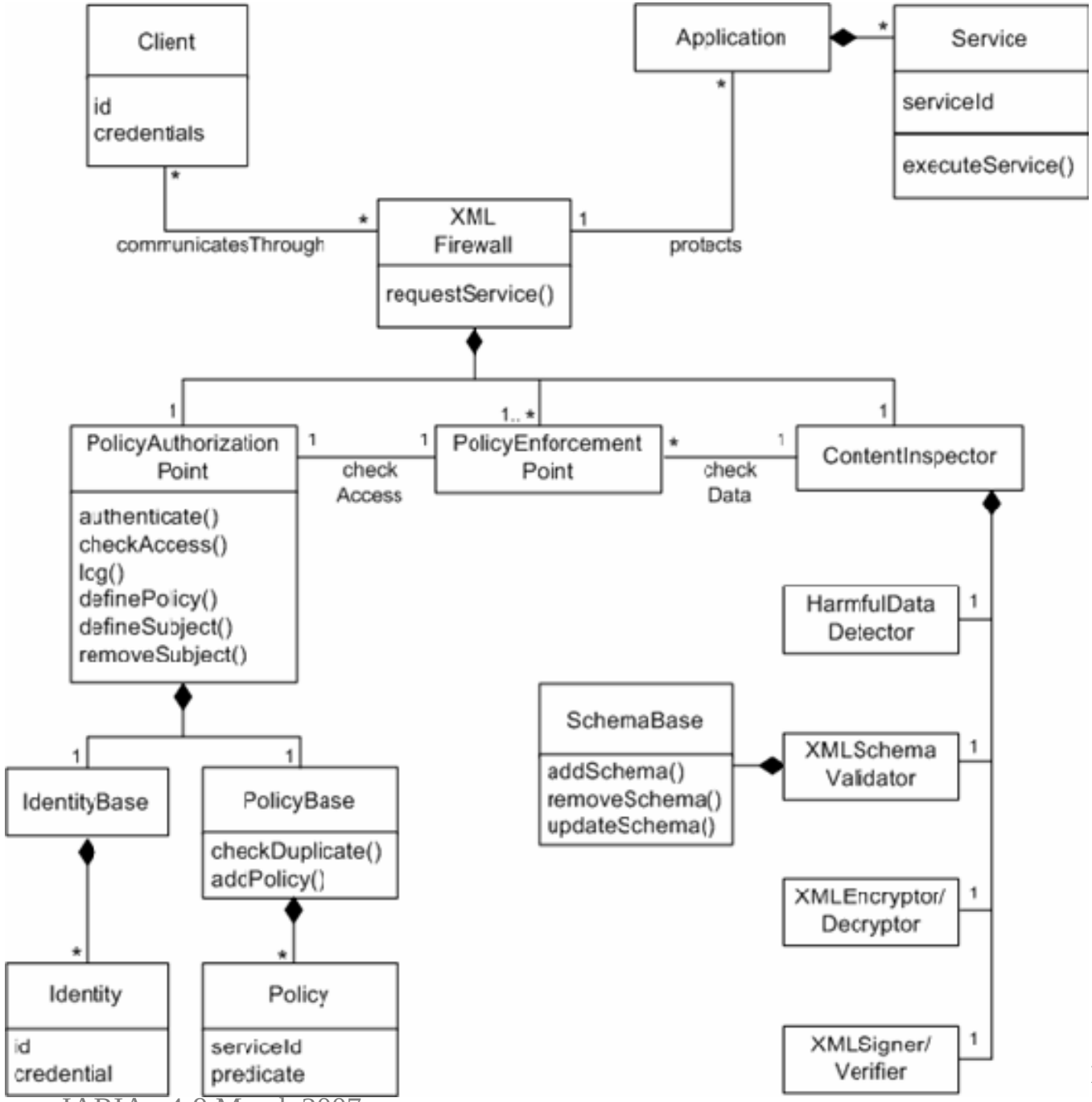
- XML firewall is a special case of it
- Controls input/output from distributed applications
- Can filter wrong commands, wrong type or length parameters, wrong sequences


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

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

Application security

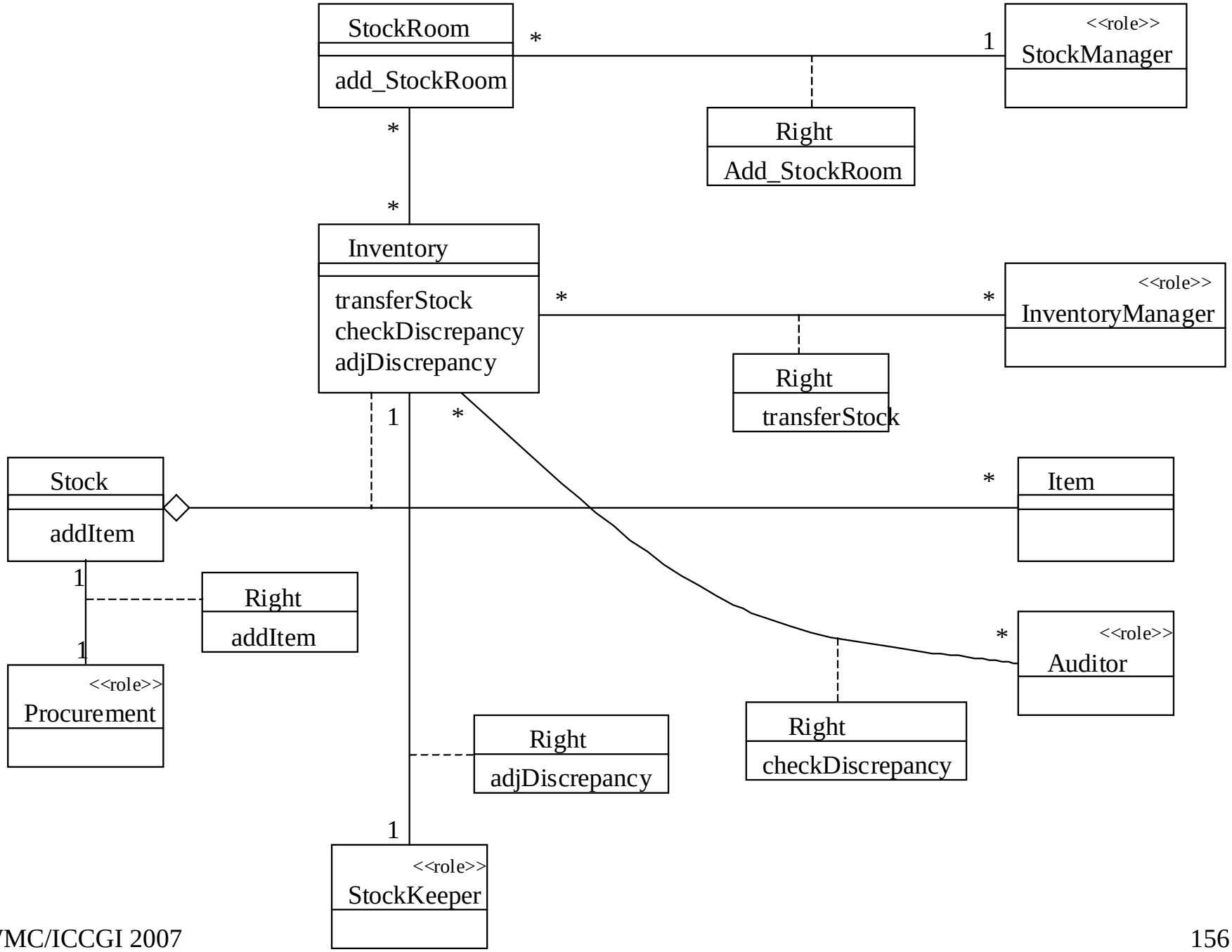
- Secure analysis patterns
- Stock manager
- Patient records
- Medical information

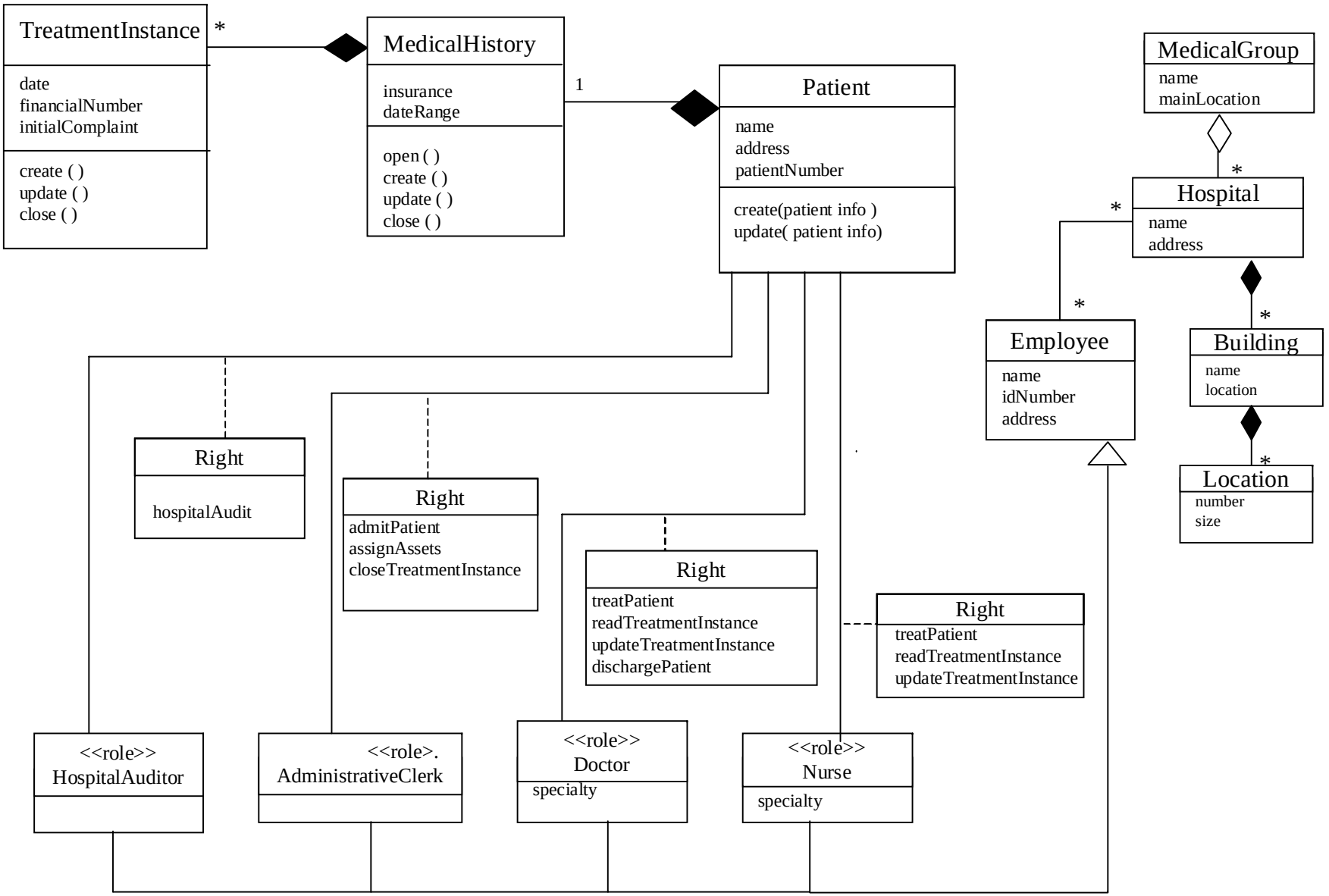
Analysis stage

Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. We can build a conceptual model where repeated applications of the Authorization pattern realize the rights determined from use cases. Analysis patterns can be built with predefined authorizations according to the roles in their use cases.

Authorized analysis patterns

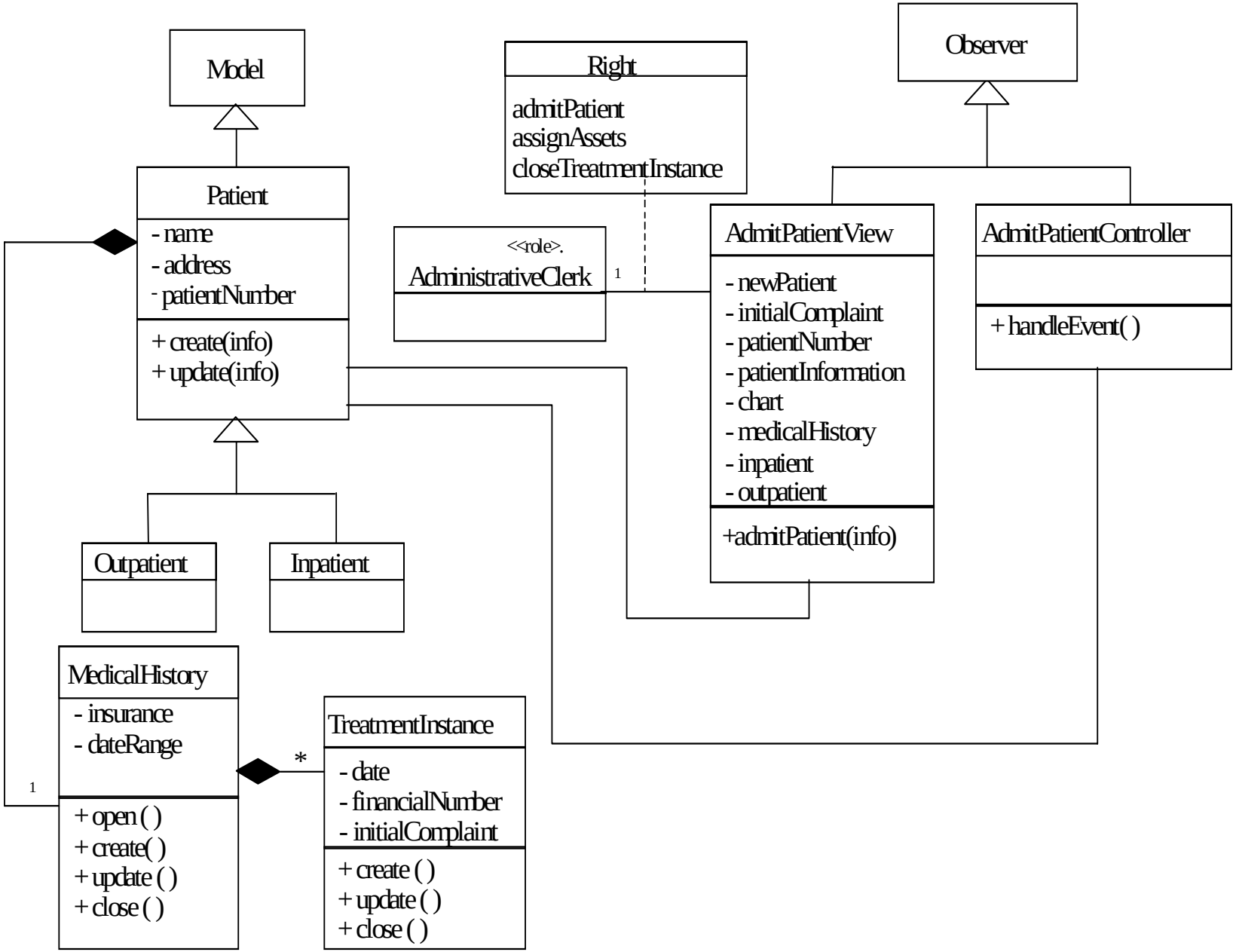
- A Sematic Analysis pattern (SAP) defines a semantic unit corresponding to a few use cases
- We can add instances of the Authorization pattern
- Examples: Authorized Stock manager, Authorized Patient Records


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


TreatmentInstance
date
financialNumber
initialComplaint
create ( )
update ( )
close ( )
*
MedicalHistory
insurance
dateRange
open ( )
create ( )
update ( )
close ( )
1
Patient
name
address
patientNumber
create(patient info)
update(patient info)
MedicalGroup
name
mainLocation
*
Hospital
name
address
*
*
Employee
name
idNumber
address
*
Building
name
location
*
Location
number
size
Right
hospitalAudit
Right
admitPatient
assignAssets
closeTreatmentInstance
Right
treatPatient
readTreatmentInstance
updateTreatmentInstance
dischargePatient
Right
treatPatient
readTreatmentInstance
updateTreatmentInstance
<<role>>
HospitalAuditor
<<role>.
AdministrativeClerk
<<role>>
Doctor
specialty
<<role>>
Nurse
specialty

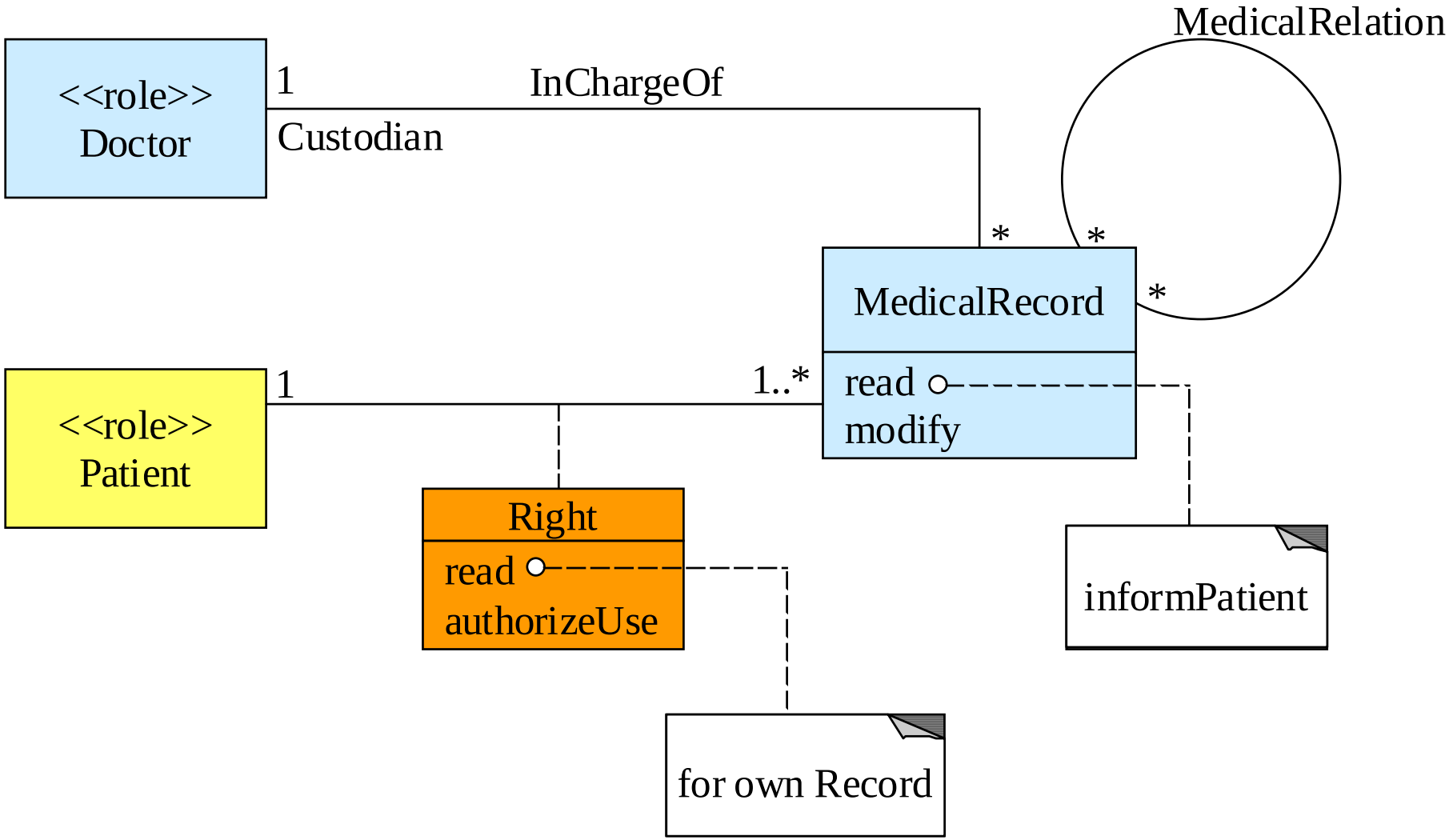
Design stage

User interfaces should correspond to use cases. Interfaces can be secured applying again the Authorization pattern. Secure interfaces enforce authorizations when users interact with the system. Finally, components can be secured by using JAAS rules defined according to the authorization rules for Java components or using .NET authorizations for .NET components. Deployment diagrams can define secure configurations to be used by security administrators.


Model
Right
admitPatient
assignAssets
closeTreatmentInstance
Observer
Patient
- name
- address
- patientNumber
+ create(info)
+ update(info)
<<role>>
AdministrativeClerk
1
AdmitPatientView
- newPatient
- initialComplaint
- patientNumber
- patientInformation
- chart
- medicalHistory
- inpatient
- outpatient
+admitPatient(info)
AdmitPatientController
+ handleEvent( )
Outpatient
Inpatient
MedicalHistory
- insurance
- dateRange
+ open ( )
+ create( )
+ update ( )
+ close ( )
1
*
TreatmentInstance
- date
- financialNumber
- initialComplaint
+ create ( )
+ update ( )
+ close ( )

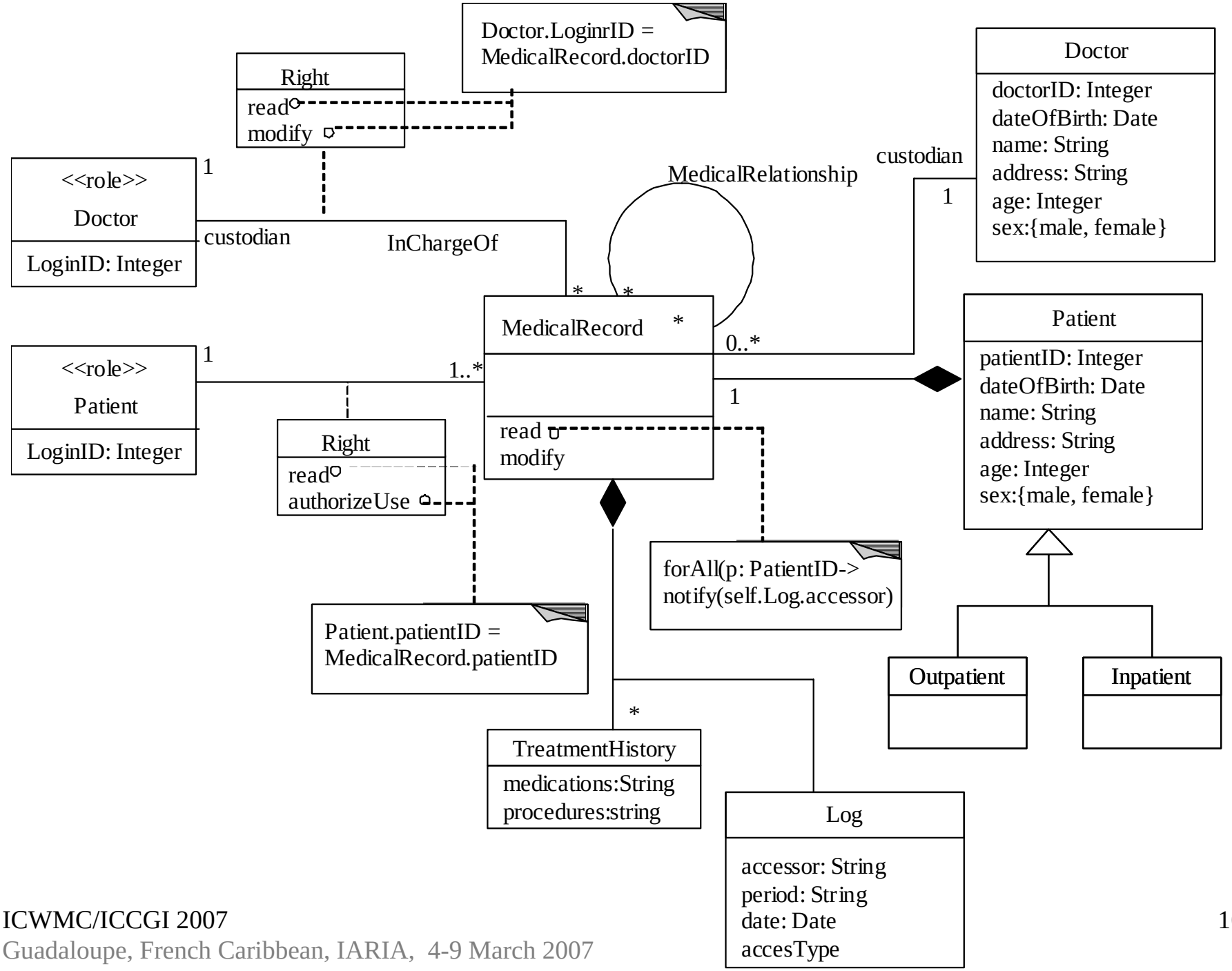
Some policies for medical information

- 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

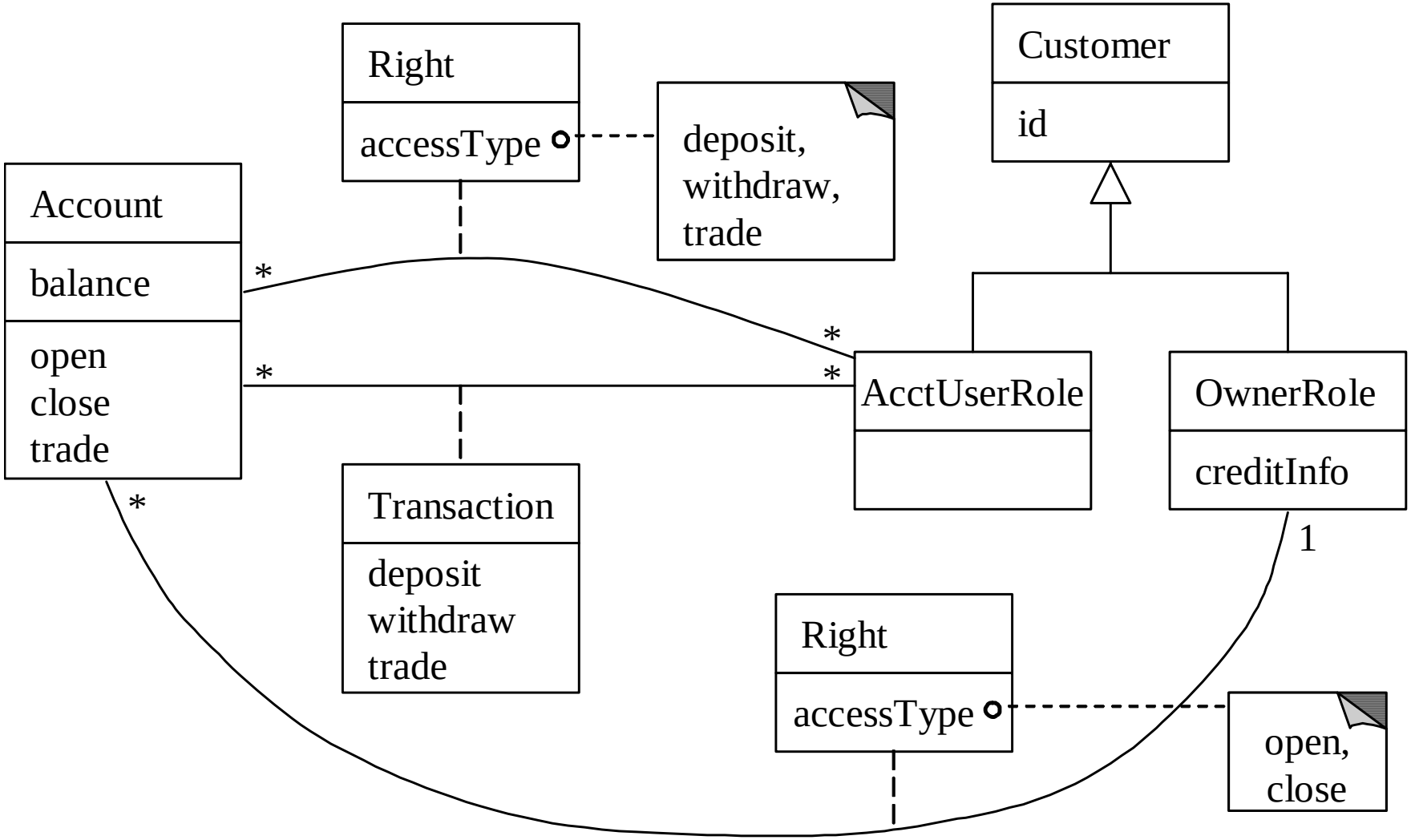

<<role>>
Doctor
1
Custodian
InChargeOf
MedicalRelation
*
*
*
MedicalRecord
read
modify
1..*
1
<<role>>
Patient
Right
read
authorizeUse
informPatient
for own Record

OCL (Object Constraint Language)

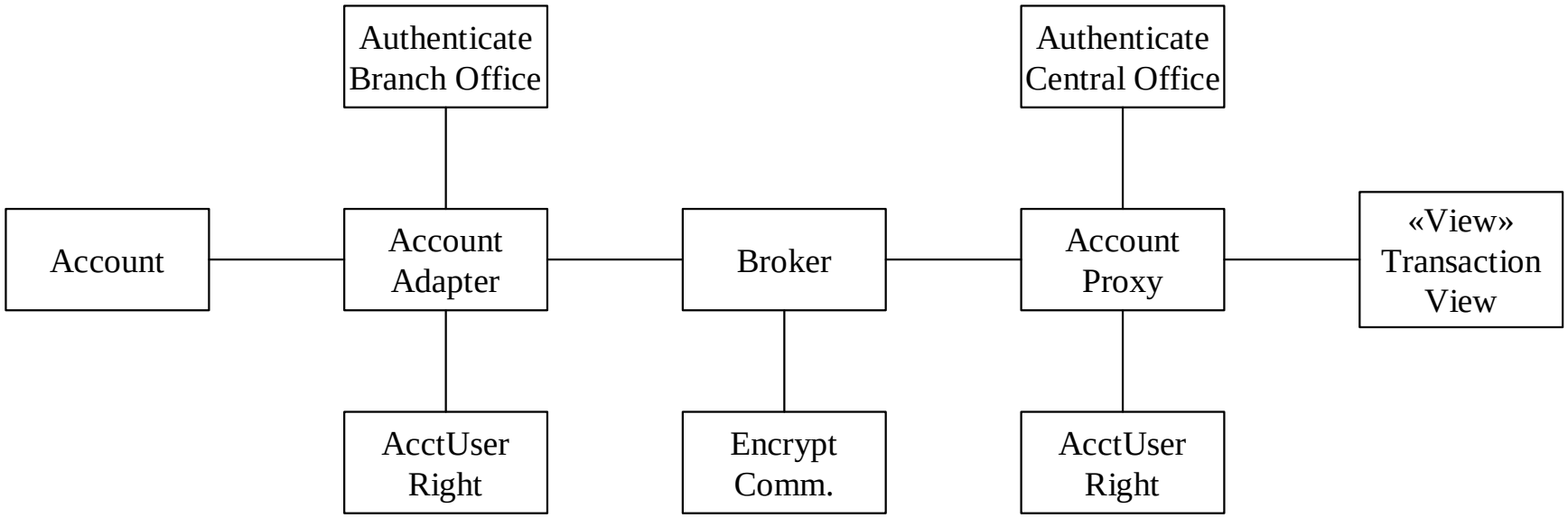
- Similar to Z and SQL, 1st order predicate calculus
- Adds precision to UML constraints
- Implementation oriented

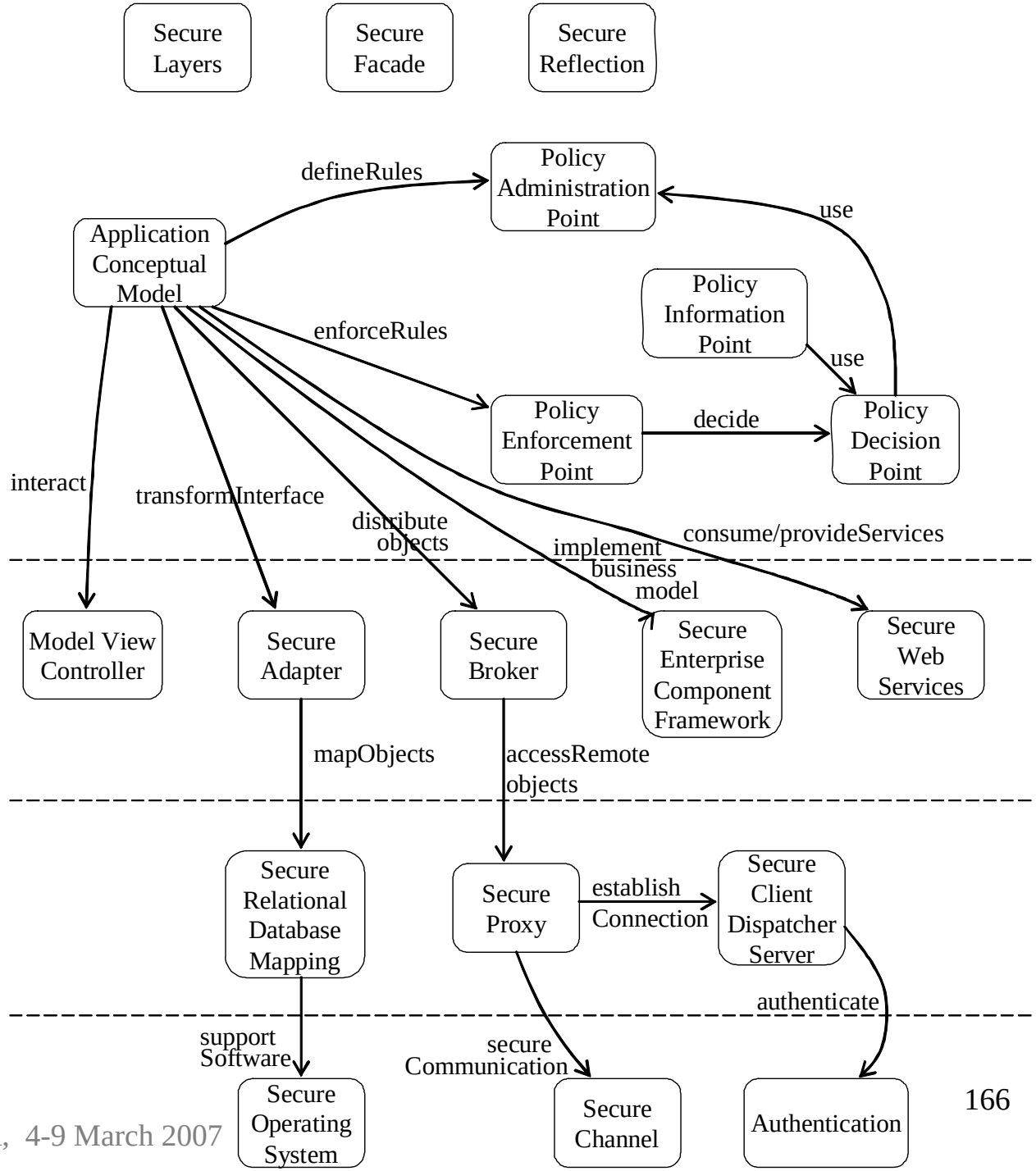

Doctor.LoginrID =
MedicalRecord.doctorID
Right
read
modify
Doctor
doctorID: Integer
dateOfBirth: Date
name: String
address: String
age: Integer
sex:{male, female}
<<role>>
Doctor
LoginID: Integer
1
custodian
InChargeOf
MedicalRelationship
custodian
1
*
*
MedicalRecord
*
0..*
<<role>>
Patient
LoginID: Integer
1
1..*
1
Patient
patientID: Integer
dateOfBirth: Date
name: String
address: String
age: Integer
sex:{male, female}
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

Rights for financial application



Design model for financial application




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

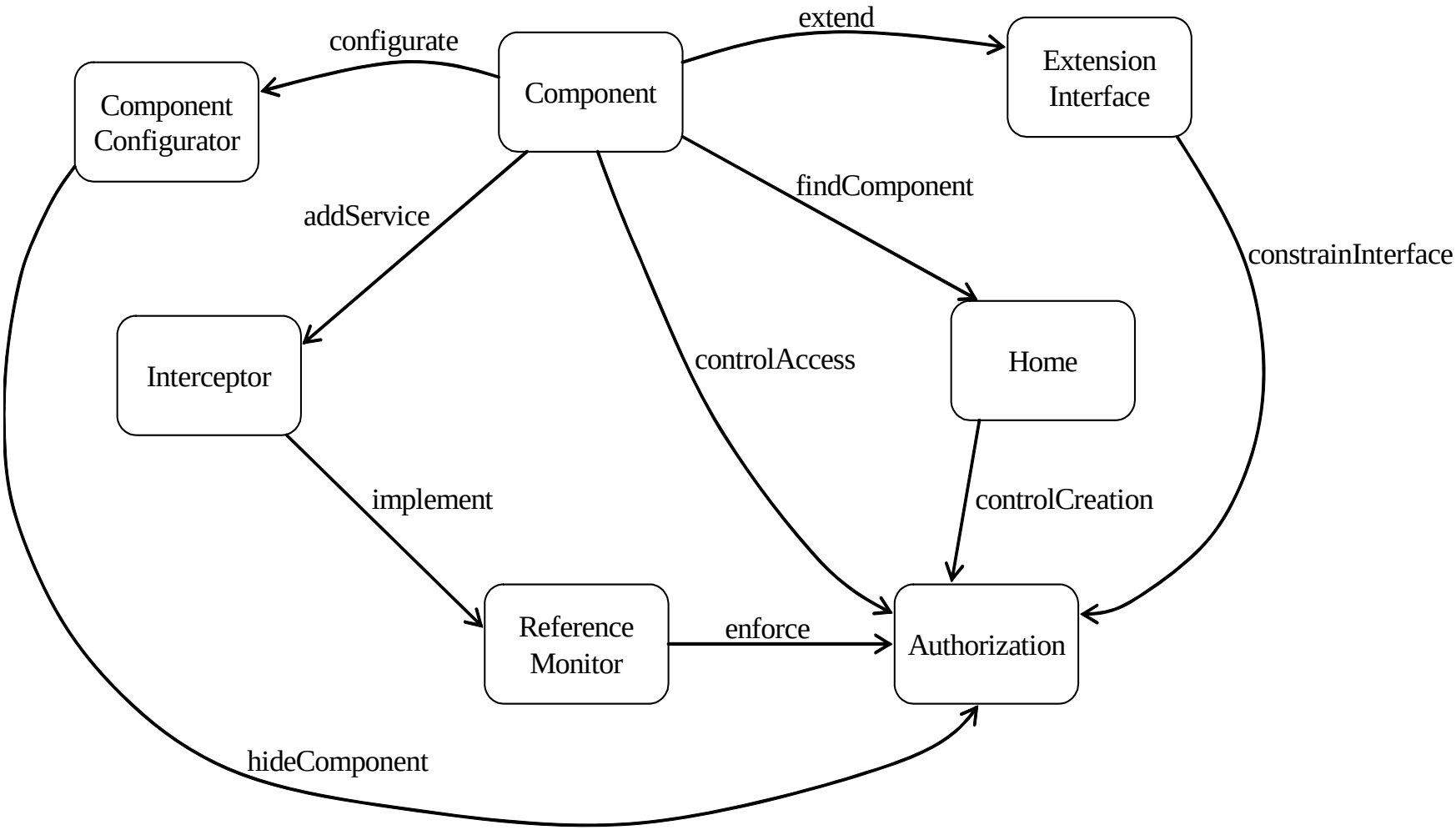
Component patterns

- The *Component Configurator* lets an application dynamically attach and detach components or processes.
- The *Interceptor* allows the transparent addition of services to an application or framework. These services are automatically invoked when certain events occur.
- The *Extension Interface* defines multiple interfaces for a component.
- The *Home pattern* separates the management of components from their use by defining an interface for creating instances of components.

Secure component patterns

- The Component Configurator can be used to reduce the time when modules are exposed to attacks. Also, modules with different degrees of security could be used in the presence of attacks or for critical applications.
- The Interceptor is useful to add security to a framework, e.g. a CORBA-based system, if the original implementation did not have it.
- The Extension Interface can be used to define views that let a user or role access only some parts of the information in specific ways, according to their authorizations.
- The Home pattern can be used to apply authorization rules to control the creation of objects in components as it has been done in operating systems.

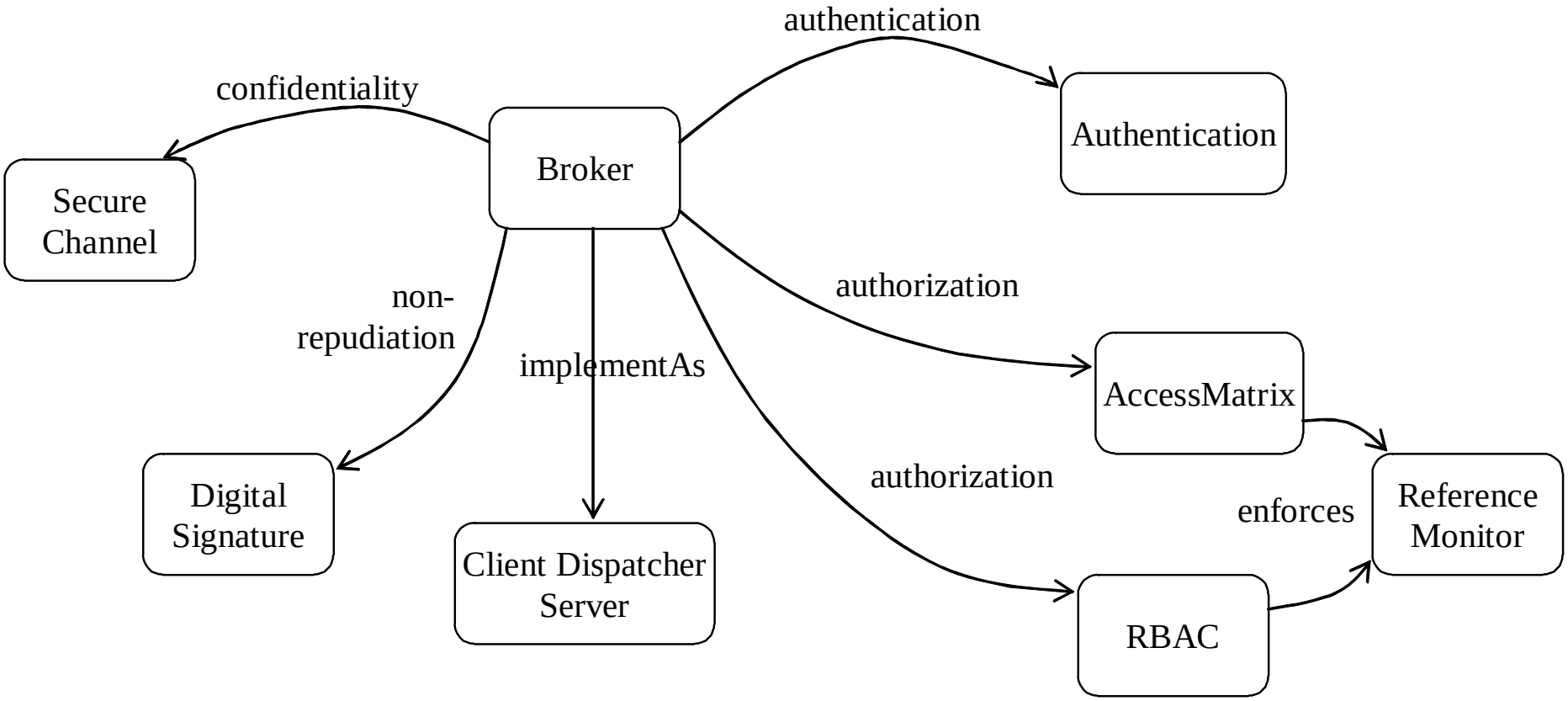
Adding security to components



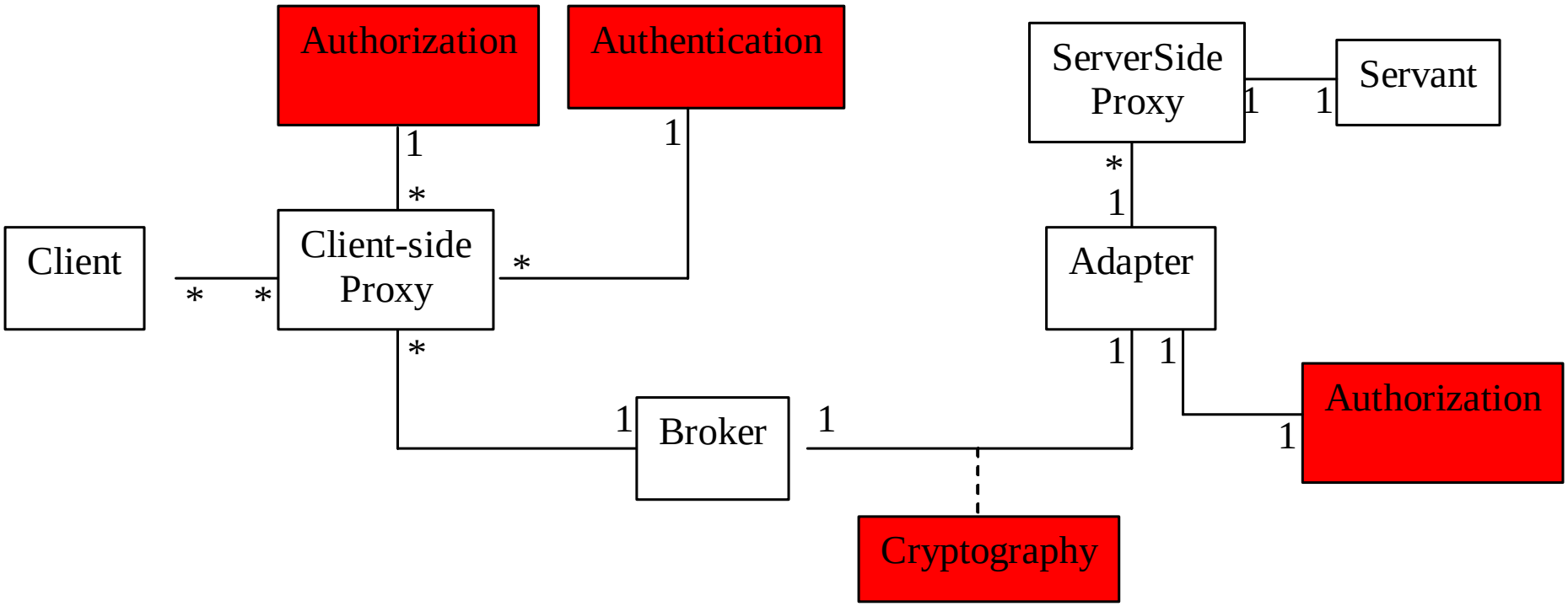
Approach

- To add security to a pattern, compose it with other patterns that correspond to appropriate security mechanisms
- The mechanisms selected depend on the expected attacks and institution policies

Adding security to the Broker

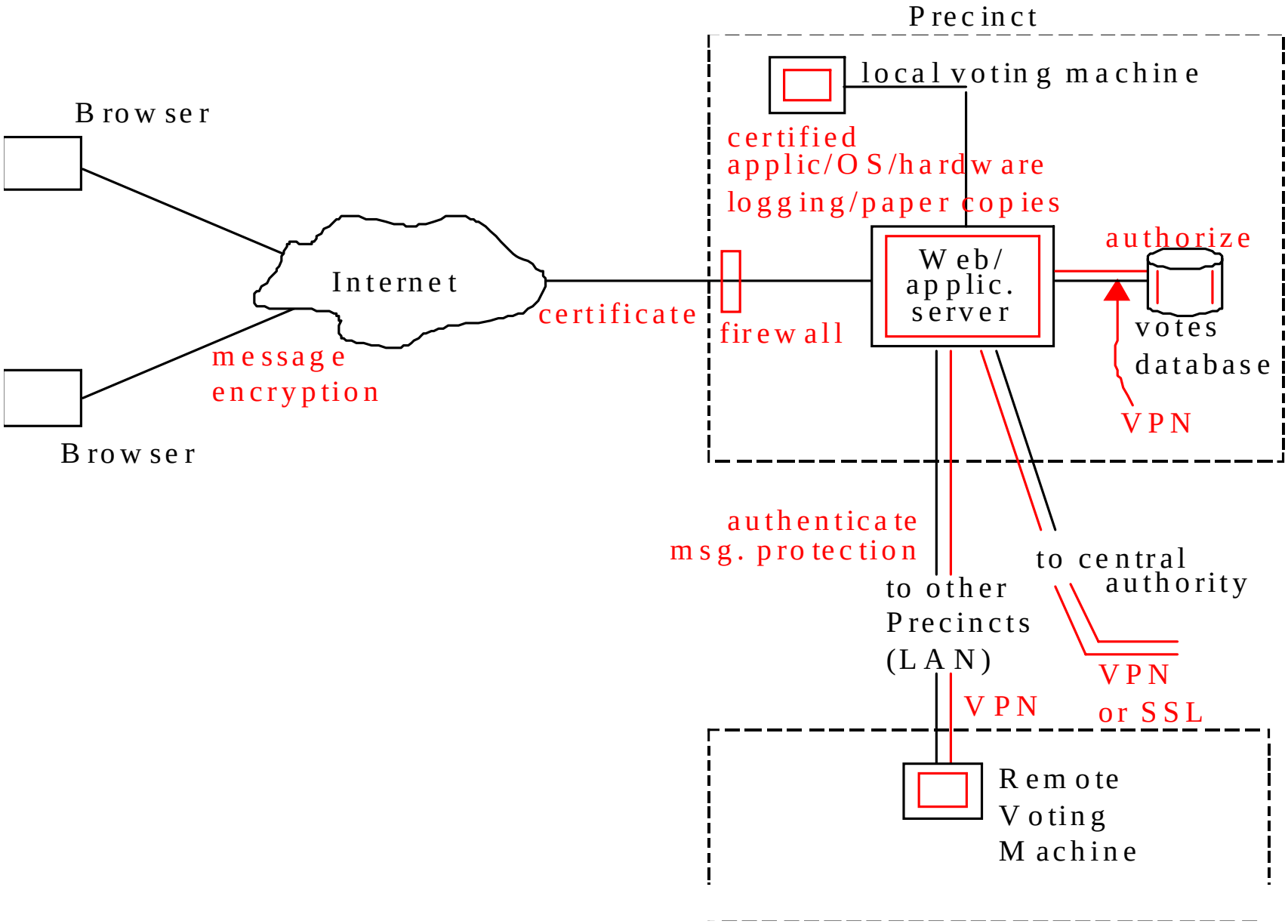


Secure Broker



Implementation stage

This stage requires reflecting in the code the security rules defined for the application. Because these rules are expressed as classes, associations, and constraints, they can be implemented as additional classes. We also need to select specific security packages, e.g., a firewall product, a cryptographic package.


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

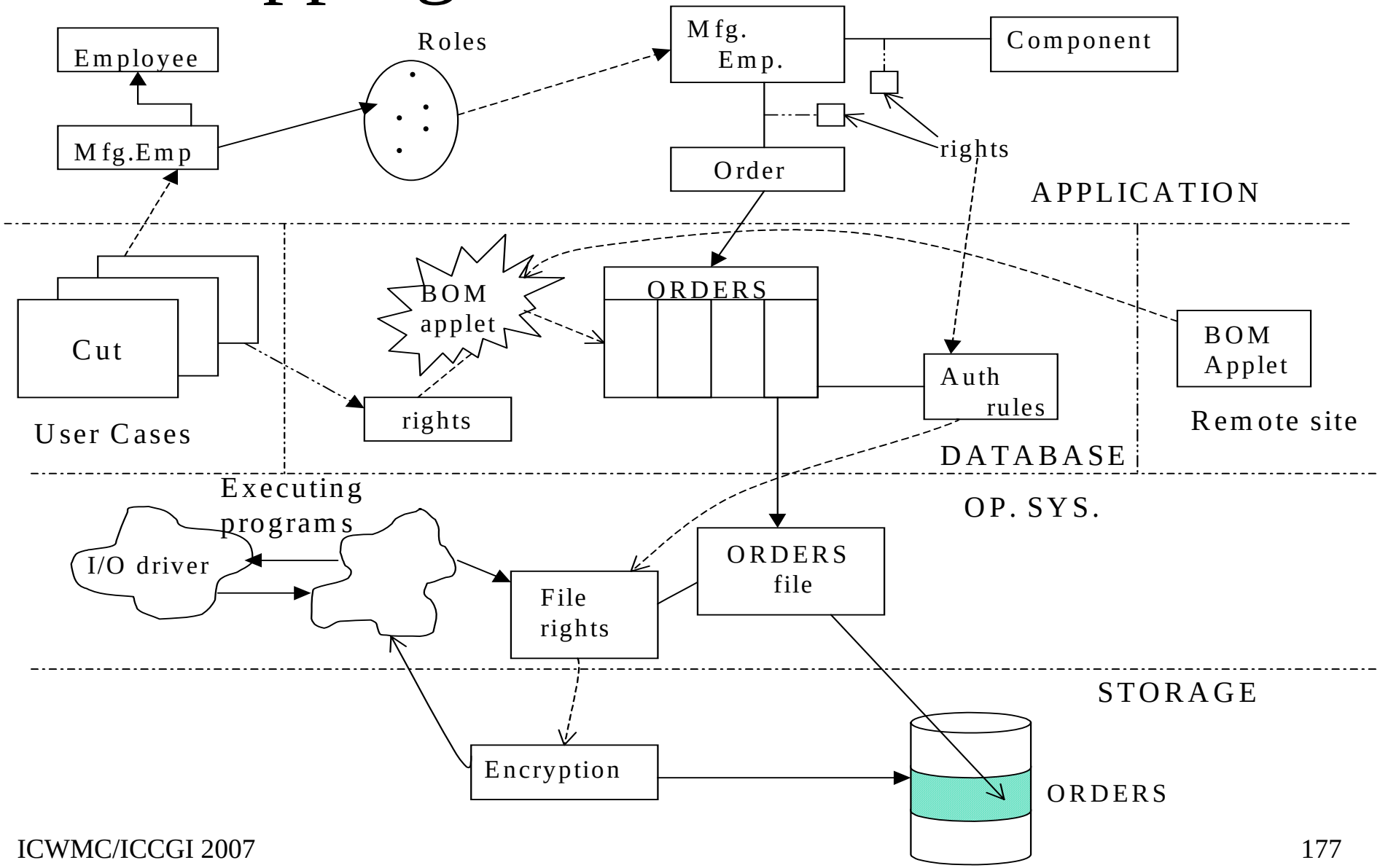
Other security patterns

- Patterns for RBAC implementation
- Cryptographic patterns (Braga, Lehtonen, Andrade)
- Java security patterns
- Single Point of Access (Joe Yoder)
- Remote Authenticator/Authorizer (EF)
- VoIP (M. Koch, EF)
- Aspect-oriented security (Rocha, Paz)

Secure architectures

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

Mapping of authorization rules



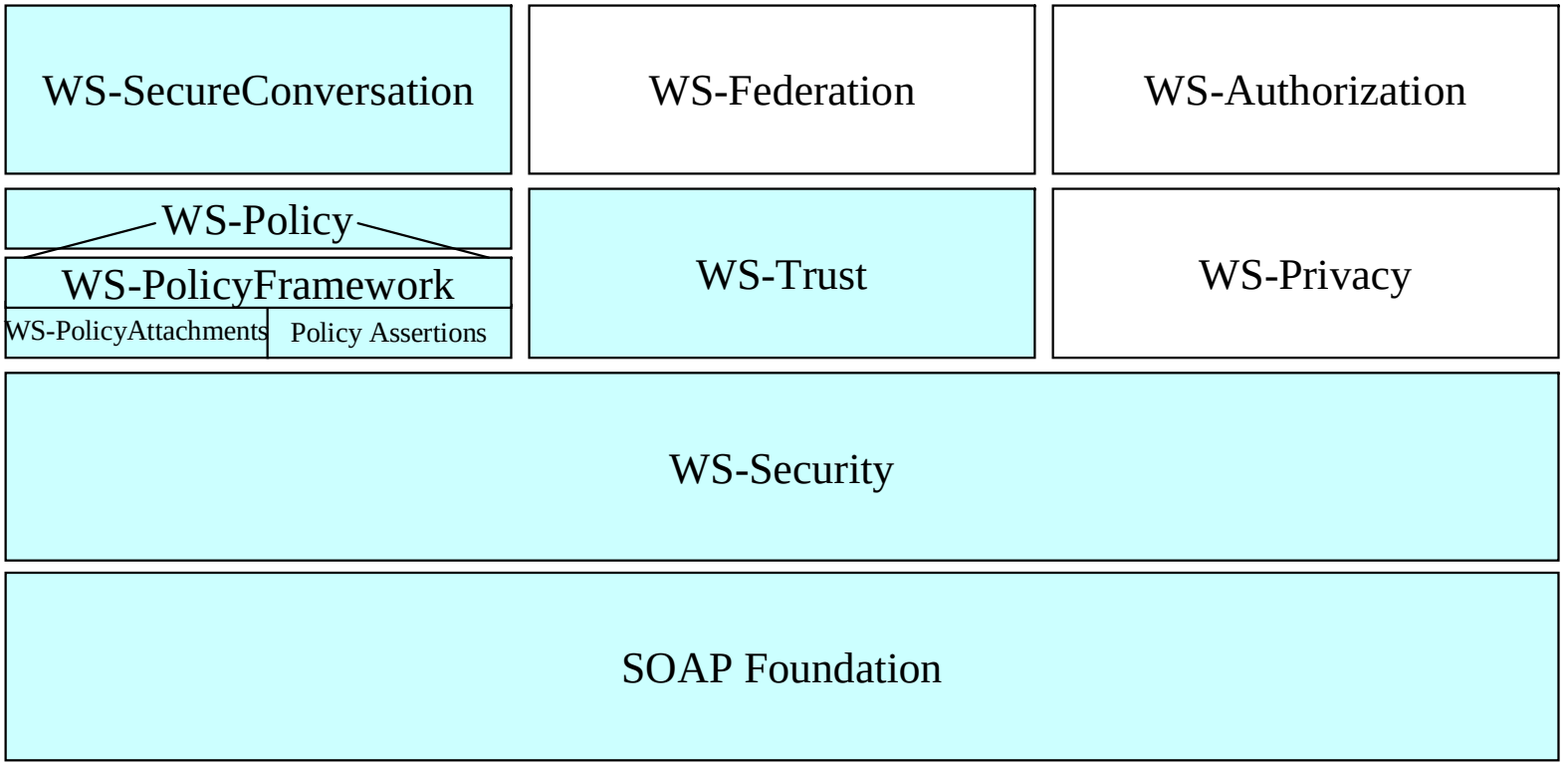
Conclusions and future work

- Internet-based systems are very flexible, but also very complex and changing
- Current 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: WS-Policy, BPEL, WSDL
- Patterns for database systems
- Combination with Aspect-Oriented Programming
- Define precise mappings between levels
- Cryptographic patterns

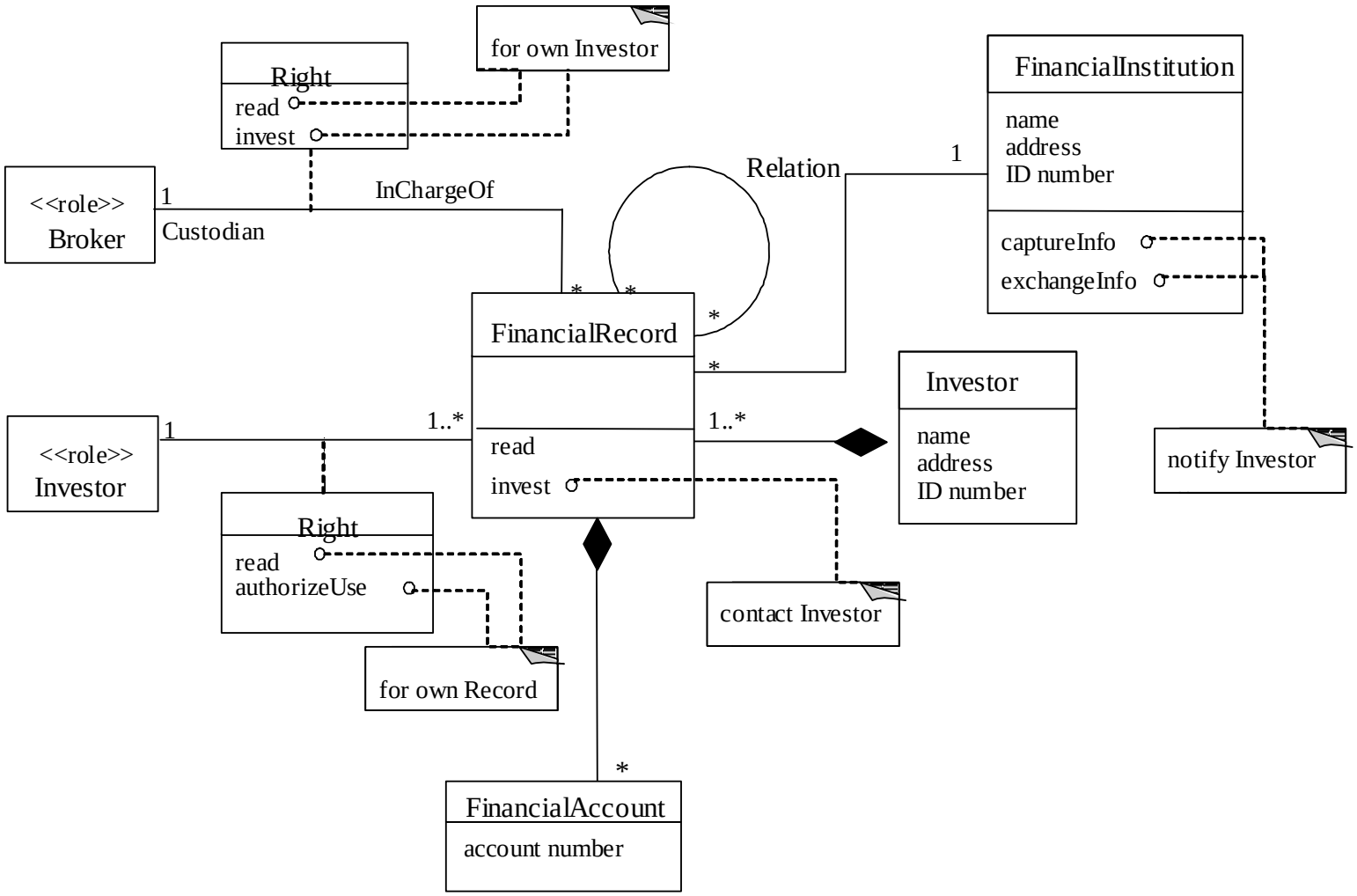
Industry standards



More research

- Refine the development method: wireless architectures
- Patterns for agent security
- Authorized analysis patterns
- Conformance of standards through patterns
- Combination of patterns and formal methods
- Secure reference models, considering all levels
- Combination with fault tolerance and real-time patterns

Sarbanes Oxley policies



Tutorial History

This tutorial has been presented at:

- IFIP WCC 1998, Vienna, Austria.
- University of Buenos Aires, Argentina. Escuela de Ciencias Informaticas (ECI), July 2003.
- IEEE Intern. Symp. on Advanced Distributed Systems (ISSADS), Guadalajara, MX, January 2005 and 2006
- IEEE Southeastcon, Fort Lauderdale, FL, April, 2005
- Third International Workshop on Security in Information Systems (WOSIS-2005), Miami, May 24-25, 2005
- 5th Latin American Conference on Pattern Languages of Programs, Campos do Jordao, Brazil, August 16-19, 2005
- IEEE Int. Symposium on Secure Software Engineering (ISSSE.06), Arlington, VA, March 2006.
- Security track of the IFIP WCC 2006 (Santiago de Chile, August 2006).
- Eigth International Symposium on System and Information Security - SSI´2006, November 08-10, 2006. http://www.ssi.org.br/english/
- IARIA's Joint Third International Conference on Wireless and Mobile Communitions (ICWM 2007) and Second International Multi-Conference on Computing in the Global Information Technology, Guadaloupe, French Carbbean, March 4-9, 2007 http://www.iaria.org
- *45th ACM Southeast Conference (ACMSE 2007)*, March 23-24, 2007, Winston-Salem, North Carolina, http://acmse2007.wfu.edu

Questions?

Dr. Eduardo B. Fernandez and Dr. Maria Larrondo Petrie
Dept. of Computer Science and Eng.
Florida Atlantic University
777 Glades Rd
Boca Raton , FL 33431
Tel. (561) 297-3466, 297-3400
Fax (561) 297-2800
ed@cse.fau.edu, petrie@fau.edu
http://www.cse.fau.edu/~security