Software Architecture

Lecture 8

What is Software Architecture?

Néstor Cataño

Innopolis University

Spring 2016

What is Software Architecture?

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them"

Slides are based on book "Software Architecture in Practice", 3rd Edition, by Kazman et al.

Consequences of the previous definitions

- **Architecture is a set of software structures**
  - They partition the system **units or modules**
    - **E.g.** modules are assigned to **teams** or to **places** in a file structure for implementation, integration and testing.
  - They are **static**: they offer a view of a system
  - They are **dynamic**: they focus on the ways elements interact in runtime

Consequences of the previous definitions

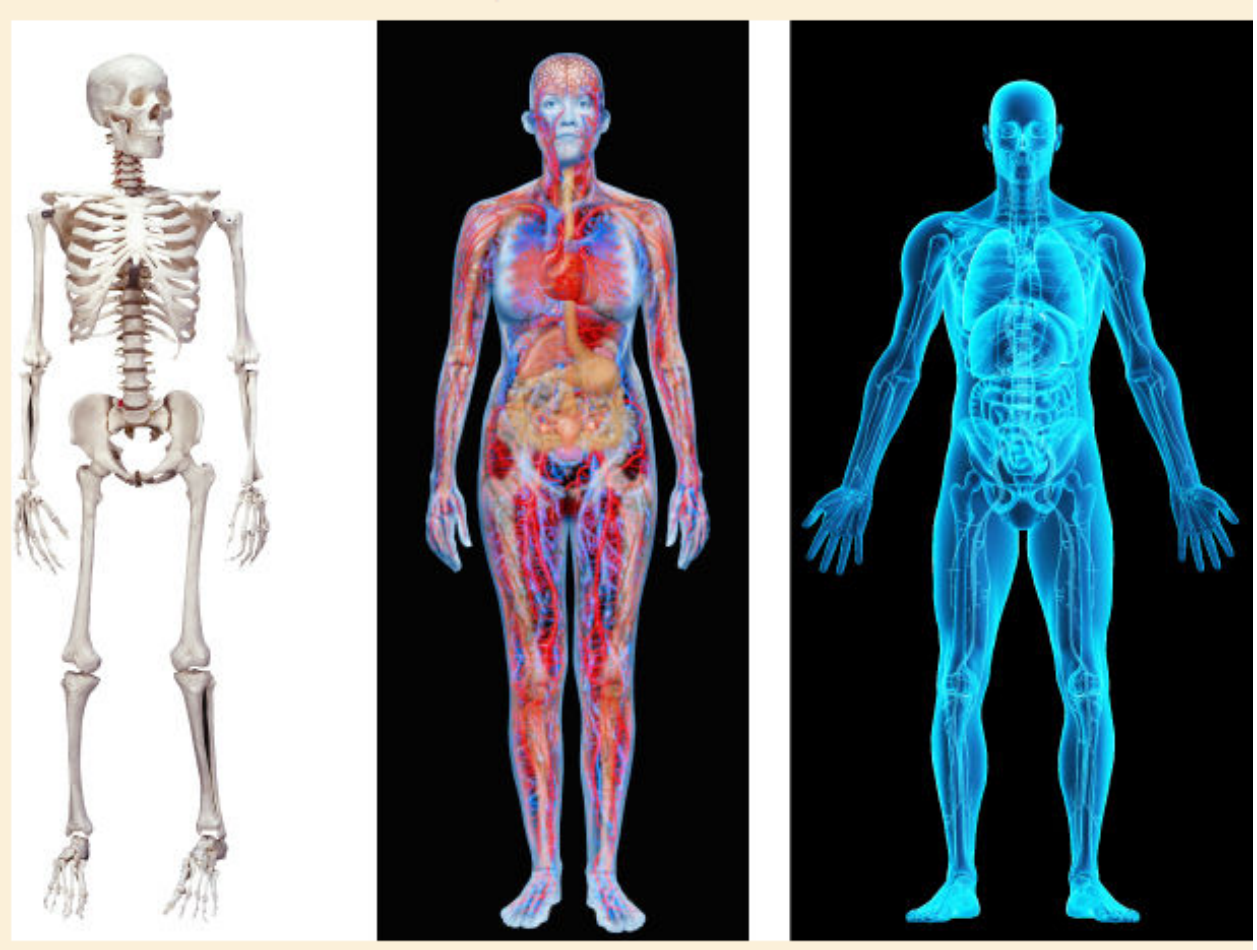
- **Architecture is an abstraction**
  - An architecture comprises software elements and how these elements relate to each other.
  - An architecture is an abstraction of a system that selects certain details and suppresses other.
- **Every software system has an architecture**
  - Every system can be shown to comprise elements and relationships among them to support some type of reasoning.
- **Architecture includes behaviour**
  - How elements interact with each other
- **Not all Architecture are Good Architecture**
  - The definition is indifferent as to whether the architecture for a system is a good one or a bad one

Architectural Structures and Views

- Neurologist, orthopedist, the hematologist, and the dermatologist all have different **views** of the structure of a human body …

- **BUT together** they describe **the architecture** of the human body

Architectural Structures and Views



Structures and Views

- **Structure** → set of elements itself as they exist in software or hardware
- **View** → representation of a coherent set of architectural elements

"Architects design structures. They document views of those structures"

Two Views of the Same Structure

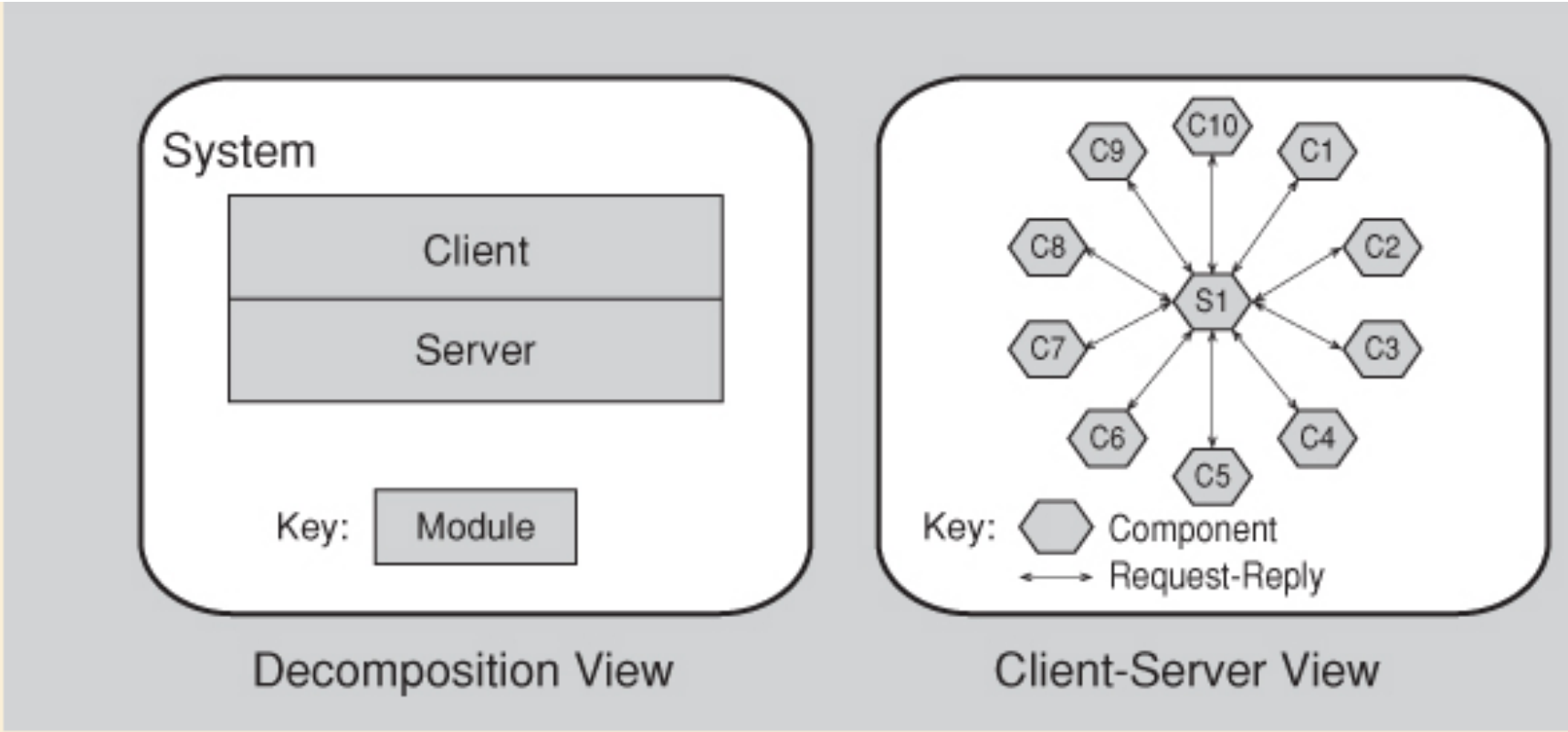


Figure 1.2. Two views of a client-server system

Architectural Drivers

- are a set of **requirements** that influence significantly your system architecture
- **can be one of the following**
  1. **Technical constraints**: technical decisions that **must be** satisfied in the **architecture**.
  2. **Business constraints**: decision imposed by business considerations that **must be** satisfied in the **architecture**.
  3. **Quality attributes**: e.g. performance, security, portability, availability, usability, etc.

Types of Structures

1. **Module structures**
2. Component-and-connector structures
3. Allocation structures

Module Structures

- Embody decisions on how the system is structured as a set of code or data units that have to be constructed or procured.
  - Elements of a module structure are **layers**, perhaps **classes**.
- Are a **static** way of representing the system

Module Structures

- **Provide answer to the following questions**
  - What's the primary functional responsibility of each module?
  - What other software elements is a module allowed to use?
  - What modules are related to others by **generalization** or **specialization** (inheritance) relations?

Useful Module Structures

- Decomposition structure
- Class structure
- Uses structure
- Layer structure
- Data model structure

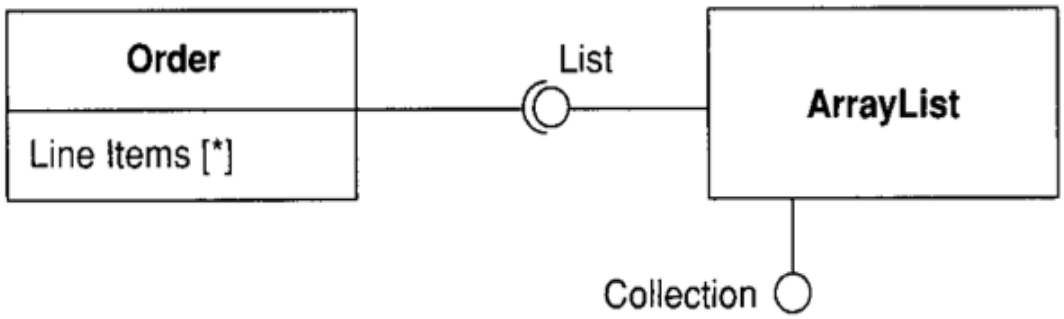
Module Structure: Decomposition Structure

- **Decomposition Structure**
  - **Units** are **modules** that are related by a **is-a-sub - module** relation
  - **Modules** often have products: interface specifications, code, test plans etc.
  - **Key Architectural Drivers** → **Modifiability** (likely changes are localized)
  - **Key UML Diagrams** → **Composite Structure Diagram, Package Diagram**

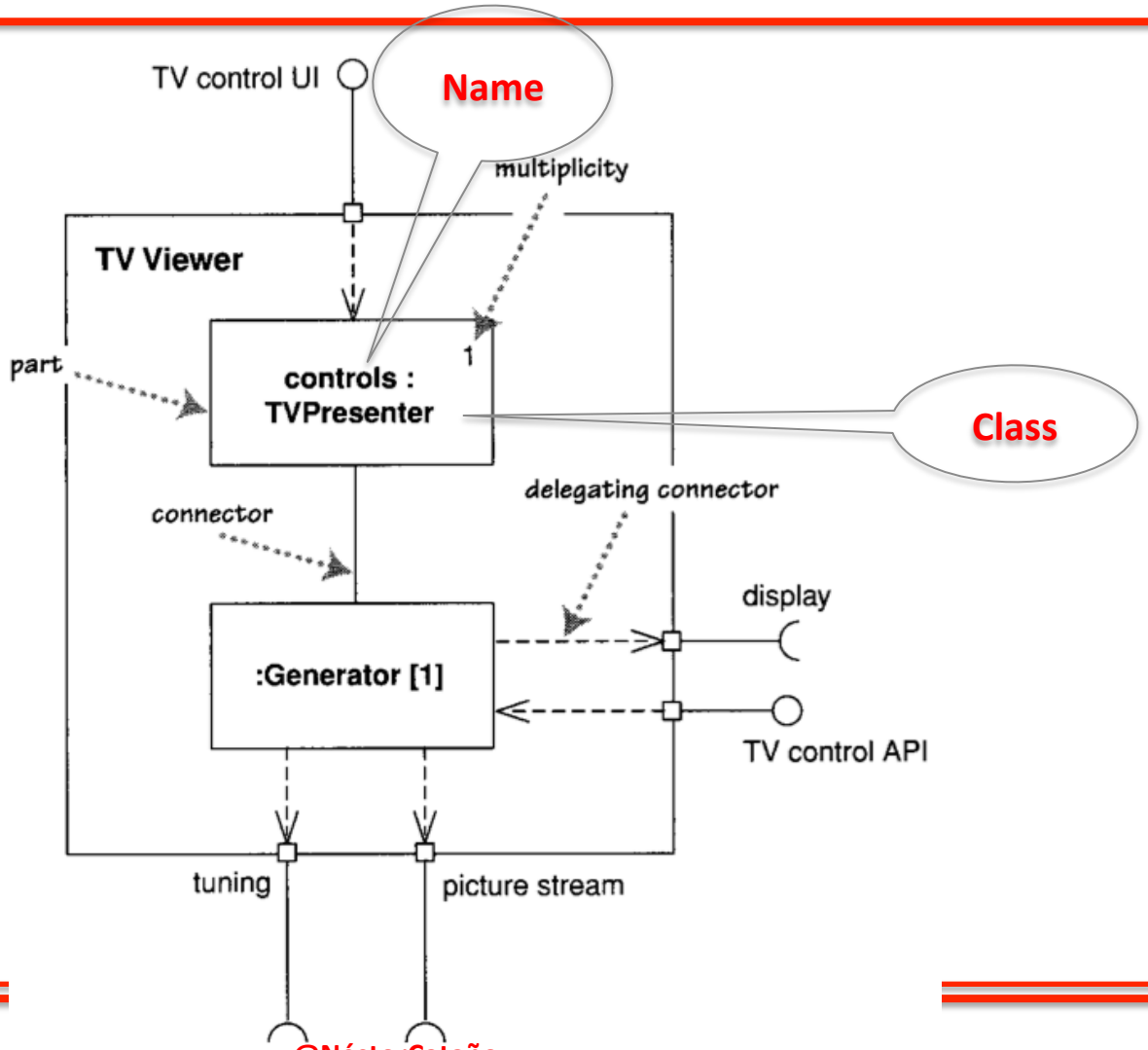
UML Composite Structures

- **UML 2** provides the ability to hierarchically decompose a class into an internal structure.
- Composite structures show the internal structure of a class.

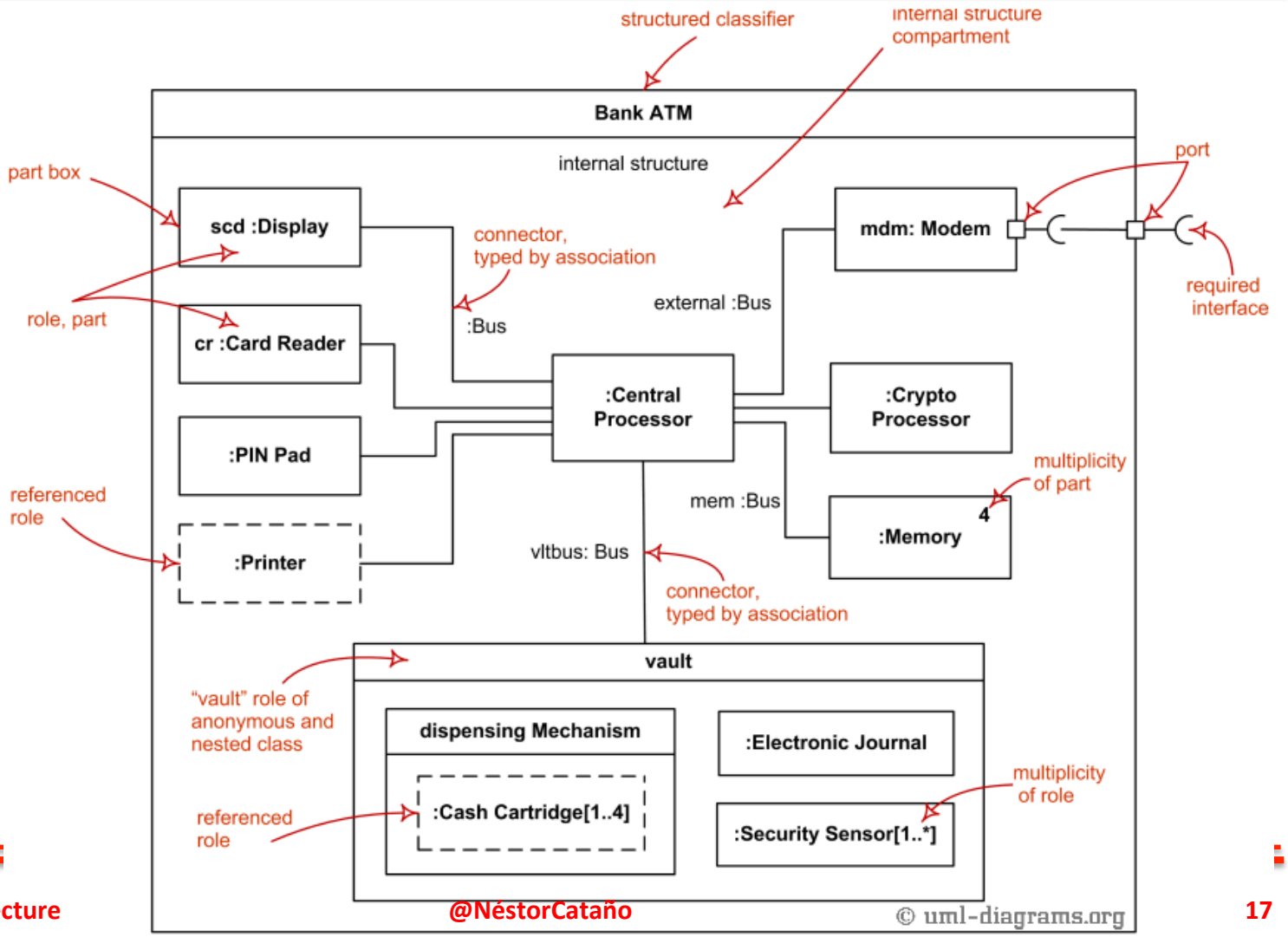
ArrayList implements a List interface, and Order requires a List interface



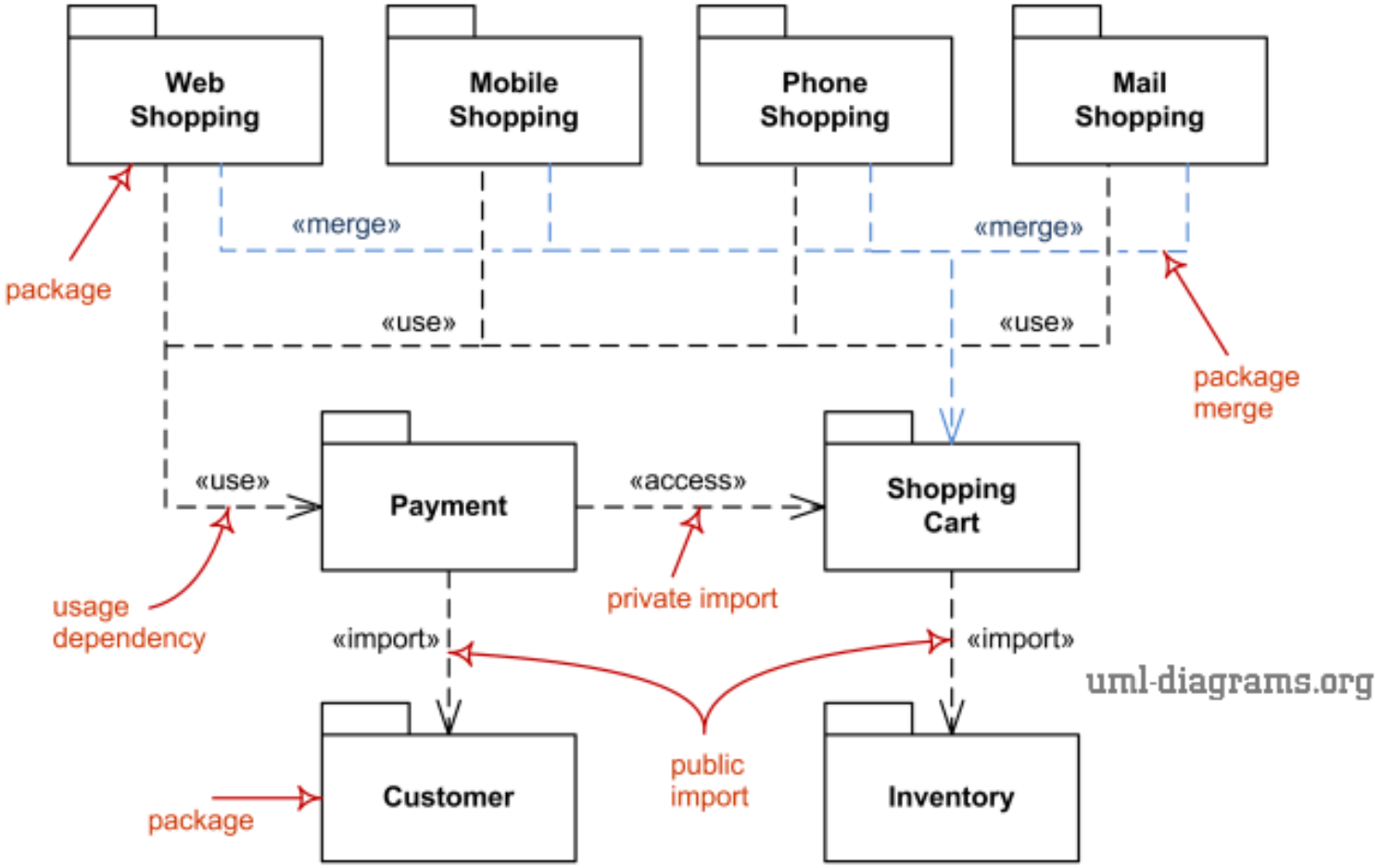
UML Composite Diagram for a TV Viewer



UML Composite Diagram for an ATM
(adapted from www.uml-diagrams.org)



Package Diagram for a Shopping System
(adapted from www.uml-diagrams.org)



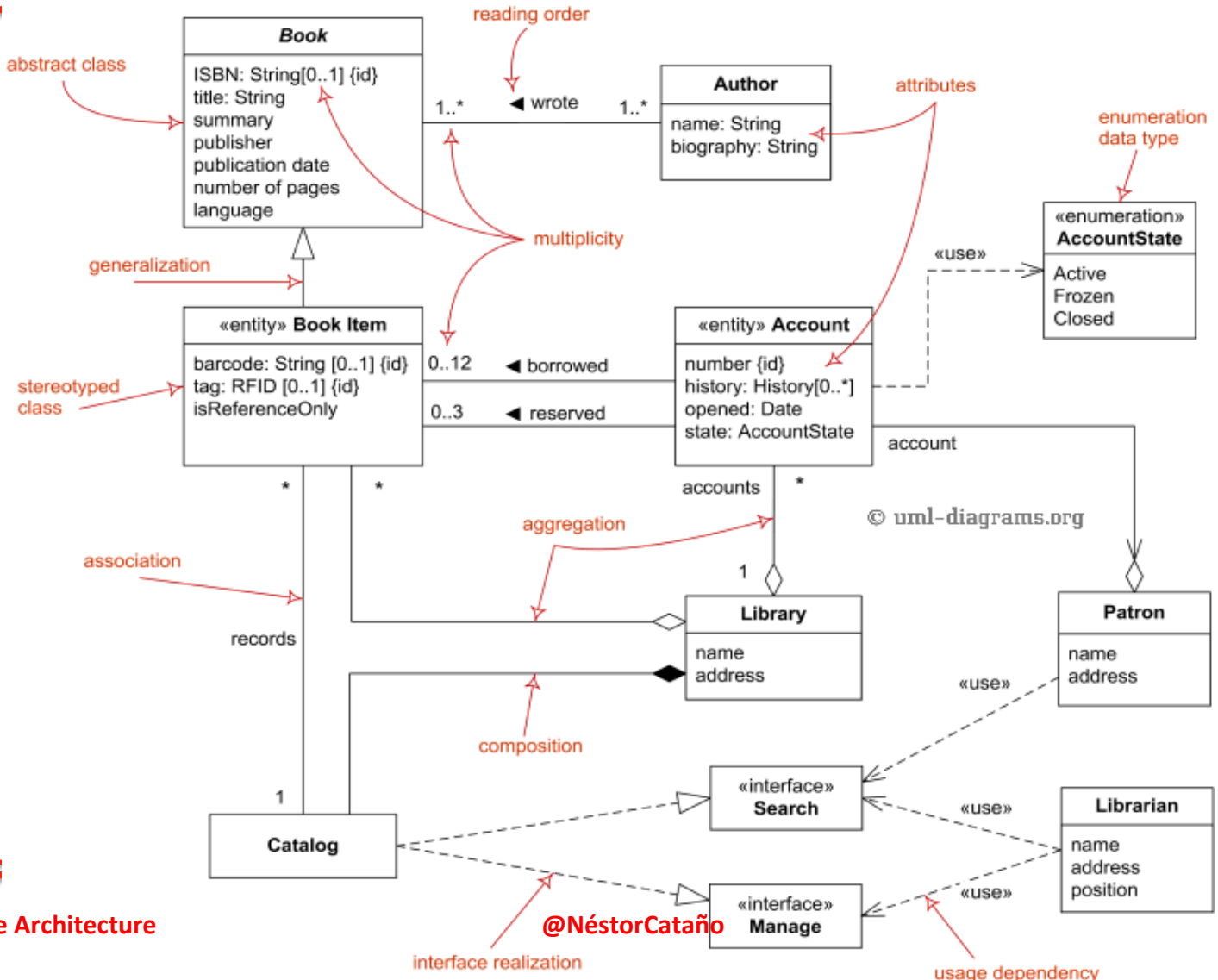
Module Structure:
Class Structure

- **Class Structure**
  - Module unites are called **classes**
  - The relation is **inherits-from** or **is-an-instance-of**
  - **Key Architectural Drivers** → **Modifiability** (likely changes are localized), **Extensibility** (ability of the system to support future changes)
  - **Key UML Diagrams** → **Class diagram**

Module Structure:
Uses Structure

- **Uses Structure**
  - Units are **modules** perhaps **classes**
  - Modules are related by a **uses relation**
  - **Key Architectural Drivers** → **Extensibility** (ability of the system to support future changes)
  - **Key UML Diagrams** → **Class Diagram**

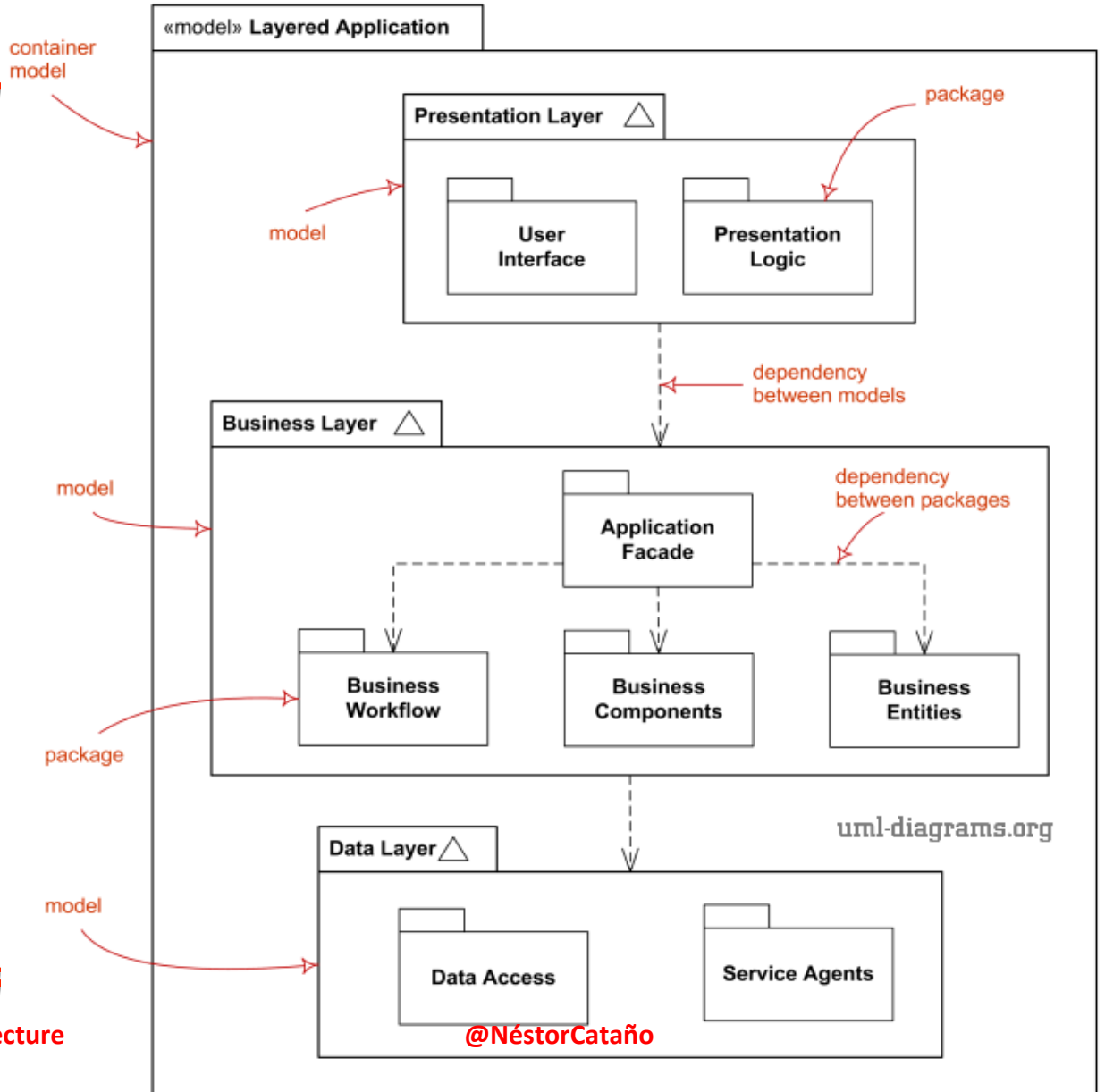
Class Diagram for A Library
(adapted from www.uml-diagrams.org)



Module Structure:
Layer Structure

- **Layer Structure**
  - Modules of this structure are **layers**
  - Modules are related by a **uses relation**
  - **Layers** are abstract virtual machine that provides a cohesive set of services through a managed interface
  - **Key Architectural Drivers** → **Portability** (the ability of a system to change the underlying computing Platform)
  - **Key UML Diagrams** → **Model Diagram**

Model Diagram



Module Structure:
Data Model Structure

- **Data Model Structure**
  - Module units are data entities and their relationships
  - E.g. for a banking system, entities would include Account, Customer, Loan, etc.
  - **Key Architectural Drivers** → **Modifiability** (likely changes are localized), **Performance** (responsiveness of a system to execute an actions)
  - **Key UML Diagrams** → **None**

Types of Structures

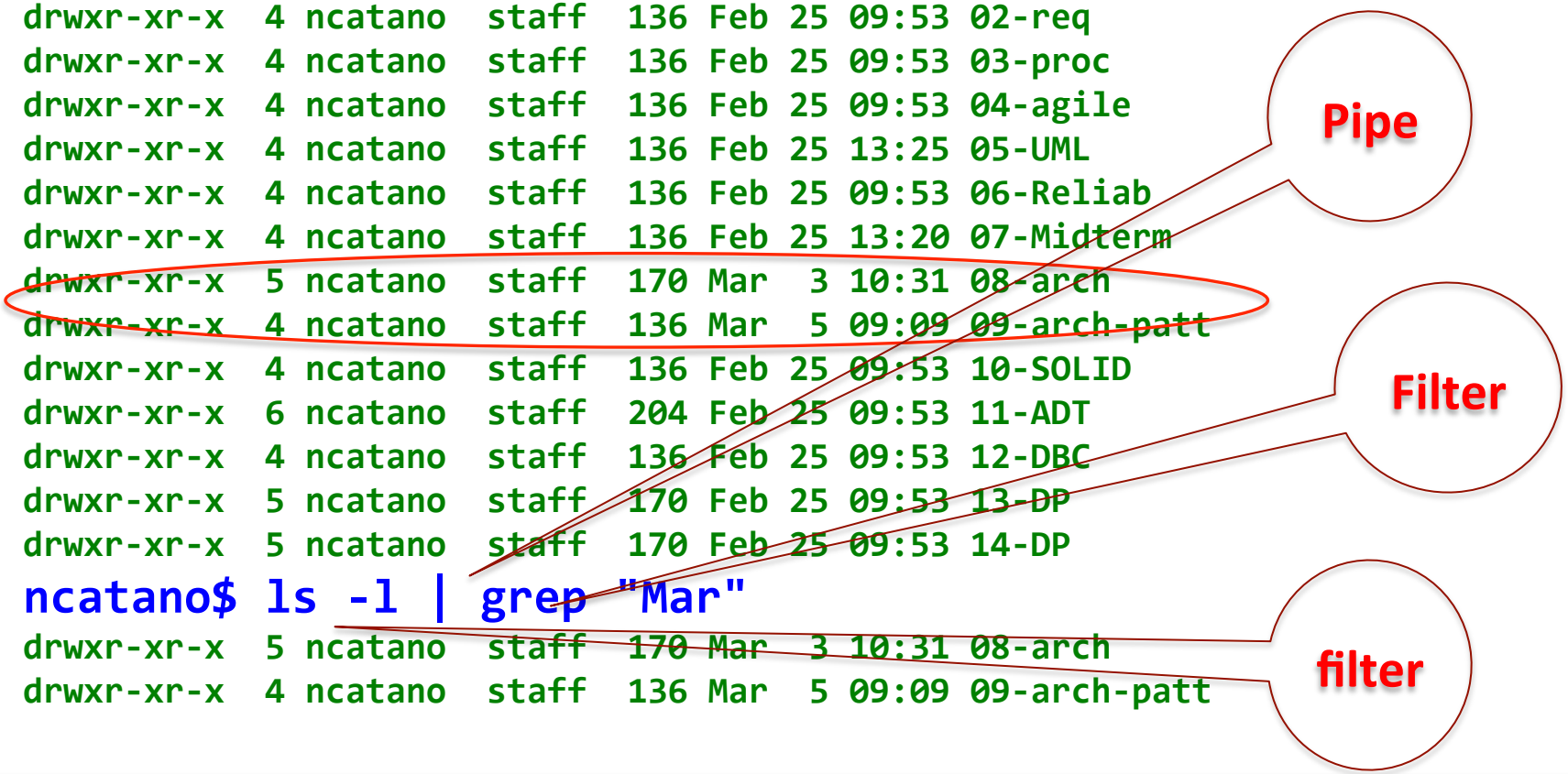
1. Module structures
2. **Component-and-connector structures**
3. Allocation structures

Component-and-Connector Structures

- **C&C Structures**
  - Embody decisions on how the system is to be structured as a set of elements that have **runtime** behavior (**components**) and interactions (**connectors**)
  - **Components** can be services, peers, clients, servers, filters, **streams**
  - **Connectors** can be call-return, process synchronization operators, **pipes**, etc.

C&C: a Unix Shell

```
ncatano$ ls -l
drwxr-xr-x  5 ncatano  staff  170 Feb 25 09:53 01-intro
drwxr-xr-x  4 ncatano  staff  136 Feb 25 09:53 02-req
drwxr-xr-x  4 ncatano  staff  136 Feb 25 09:53 03-proc
drwxr-xr-x  4 ncatano  staff  136 Feb 25 09:53 04-agile
drwxr-xr-x  4 ncatano  staff  136 Feb 25 13:25 05-UML
drwxr-xr-x  4 ncatano  staff  136 Feb 25 09:53 06-Reliab
drwxr-xr-x  4 ncatano  staff  136 Feb 25 13:20 07-Midterm
drwxr-xr-x  5 ncatano  staff  170 Mar  3 10:31 08-arch
drwxr-xr-x  4 ncatano  staff  136 Mar  5 09:09 09-arch-patt
drwxr-xr-x  4 ncatano  staff  136 Feb 25 09:53 10-SOLID
drwxr-xr-x  6 ncatano  staff  204 Feb 25 09:53 11-ADT
drwxr-xr-x  4 ncatano  staff  136 Feb 25 09:53 12-DBC
drwxr-xr-x  5 ncatano  staff  170 Feb 25 09:53 13-DP
drwxr-xr-x  5 ncatano  staff  170 Feb 25 09:53 14-DP
ncatano$ ls -l | grep "Mar"
drwxr-xr-x  5 ncatano  staff  170 Mar  3 10:31 08-arch
drwxr-xr-x  4 ncatano  staff  136 Mar  5 09:09 09-arch-patt
```



Component-and-Connector Structures

- **Provide answer to the following questions**
  - What are the major executing components and how do they interact at runtime?
  - What are the major shared data stores?
  - Which parts of the system are replicated?
  - How does data progress through the system?
  - What parts of the system can run in parallel?
  - Can the system's structure change as it executes and, if so, how?

Useful C&C Structures

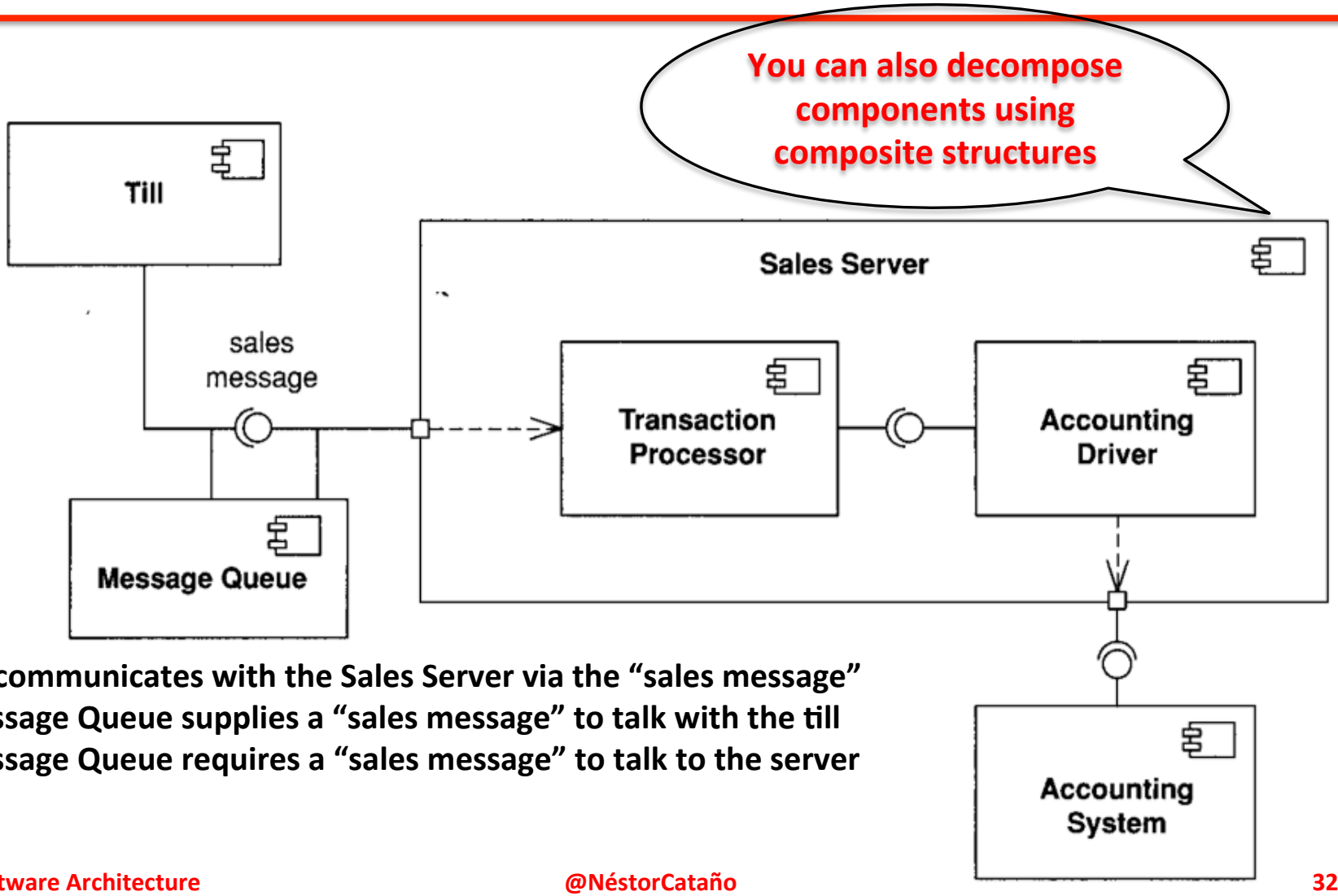
- Service structure
- Concurrency structure

C&C Structures:
Service Structure

- **Service structure**
  - Units are **services** that interoperate with each other by service coordination mechanisms such as **SOAP (Simple Object Access Protocol)**
  - **Key Architectural Drivers** → **Modifiability** (likely changes are localized), **Interoperability** (ability of a system to communicate with other systems)
  - **Key UML Diagrams** → **Component Diagram**

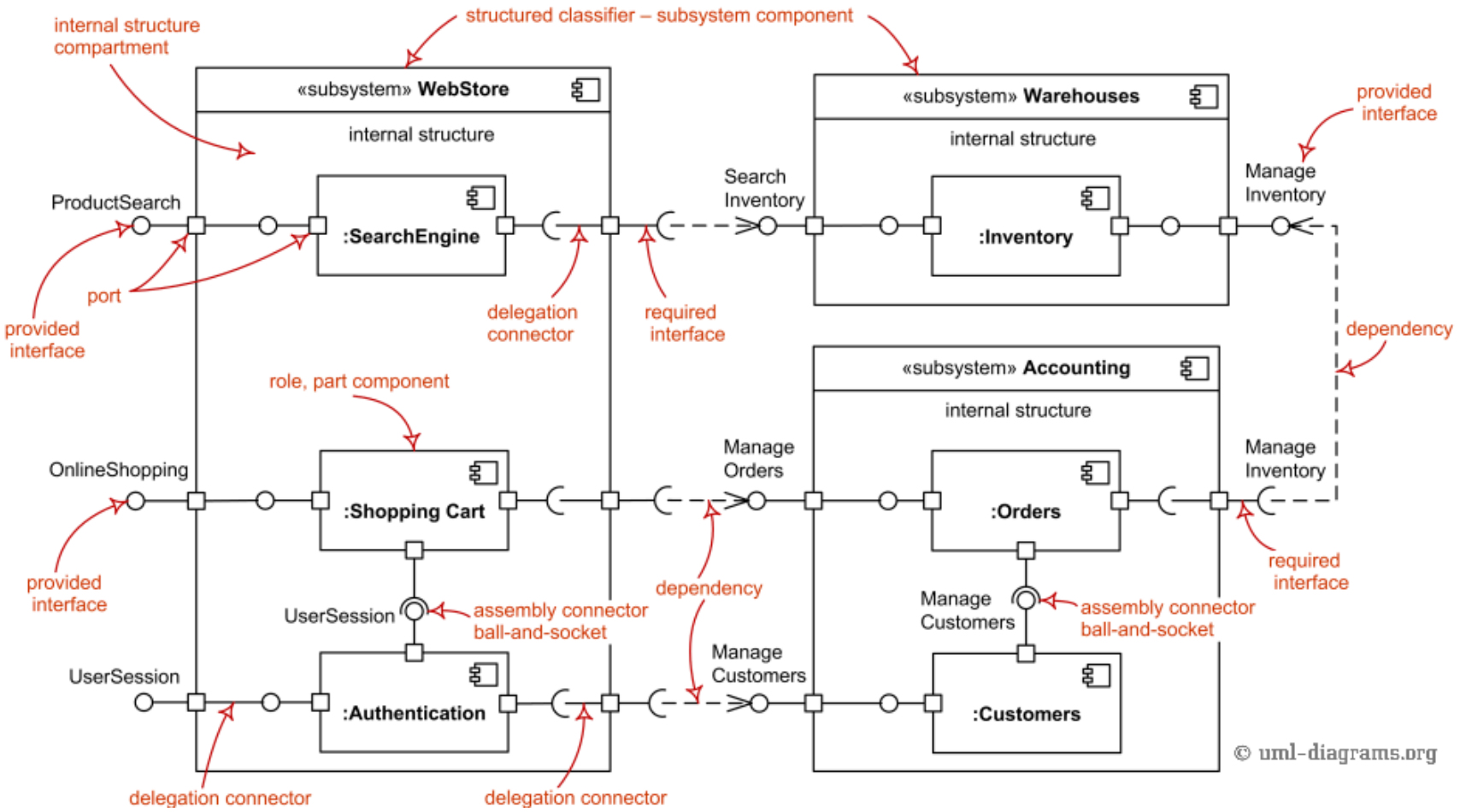
Notation for Components

A JML Component Diagram for a Sales Till connecting to a Sales Server



The Till communicates with the Sales Server via the "sales message"
The Message Queue supplies a "sales message" to talk with the till
The Message Queue requires a "sales message" to talk to the server

Component Diagram for a Sales System
(adapted from www.uml-diagrams.org)

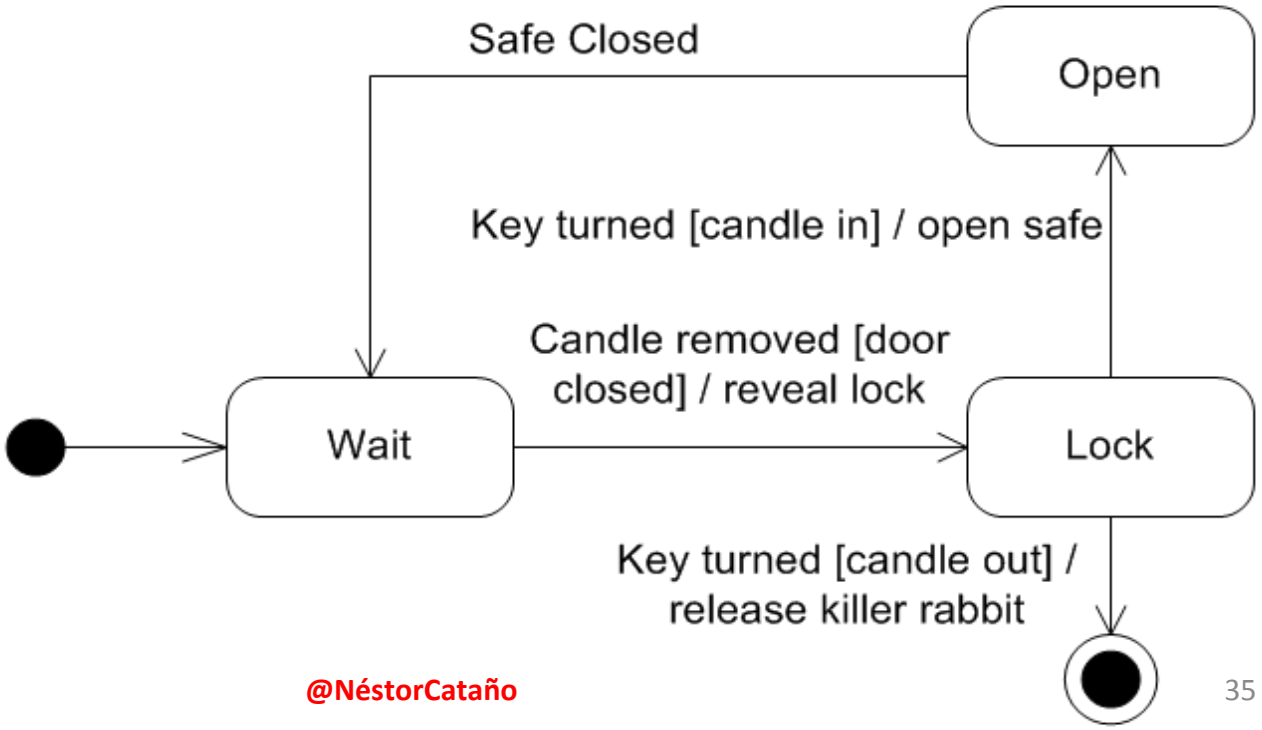


C&C Structures:
Concurrency Structure

- **Concurrency structure**
  - Allows the architect to determine opportunities for **parallelism**.
  - The **units** are **components** and the **connectors** are **communication mechanisms**
  - Components are arranged into **logical threads**
  - **Key Architectural Drivers** → **Performance** (responsiveness of a system to execute an action), **Availability** (proportion of time a system is working)
  - **Key UML Diagrams** → **Activity diagrams, state machine diagrams, sequence diagrams**

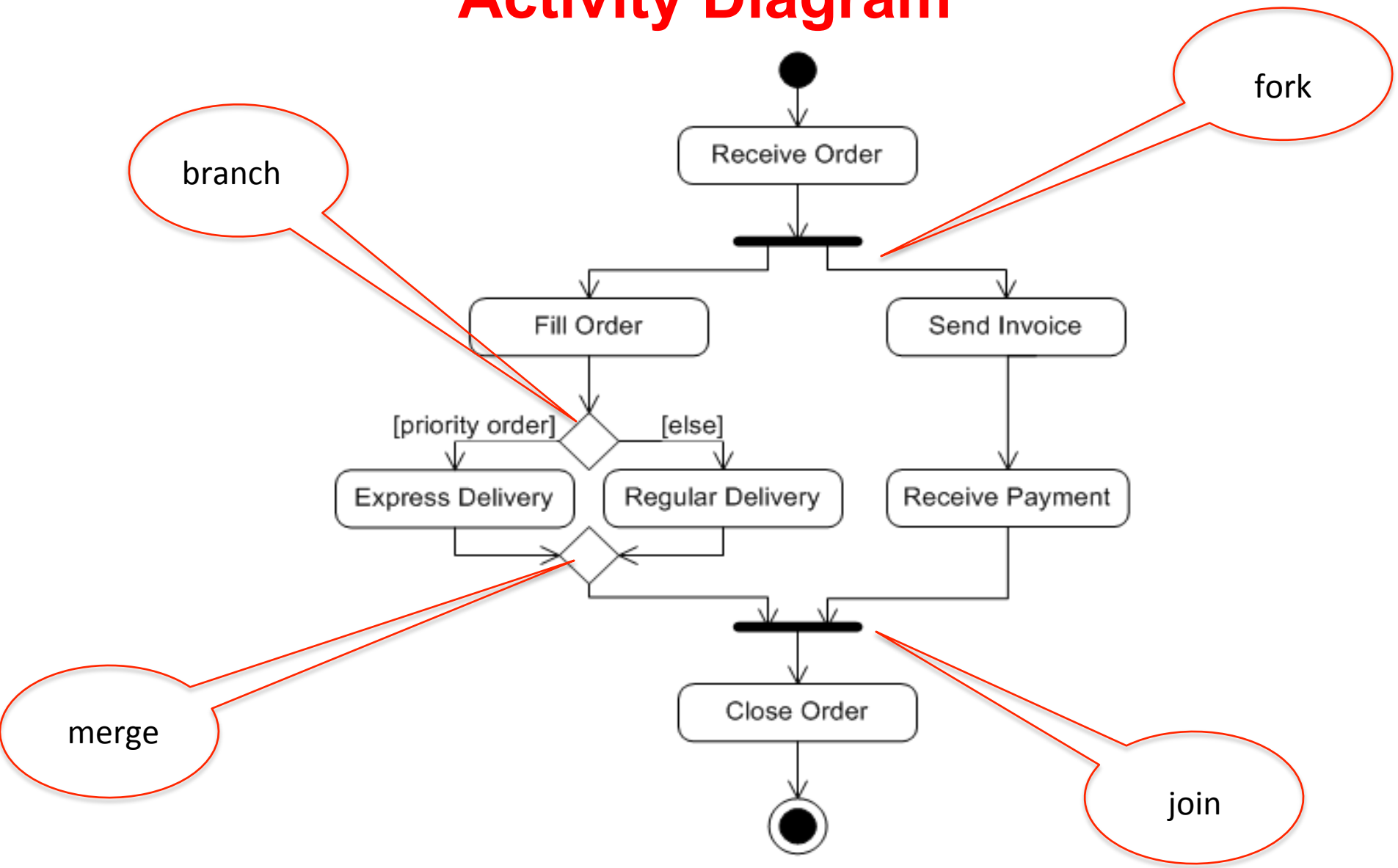
State Machine Diagram Example

1. Mysterious Castle Secret Safe and Monster! *
2. Reveal lock when special candle removed & door closed
3. Can insert key into lock when lock revelead
4. For extra safety, must replace candle before turning key
5. If candle replaced, open safe
6. If candle not replaced, release killer rabbit!



*From UML Distilled

Activity Diagram



Types of Structures

1. Module structures
2. Component-and-connector structures
3. **Allocation structures**

Allocation Structures

- Embody decisions on how the system will relate to **non-software structures** in its environment (such as **CPUs**, **file systems**, **networks**, **development teams**, etc.)

Allocation Structures

- **Provide answer to the following questions**
  - What processor does each software element execute on?
  - In what directories or files is each element stored during development, testing, and system building?
  - What is the assignment of each software element to development teams?

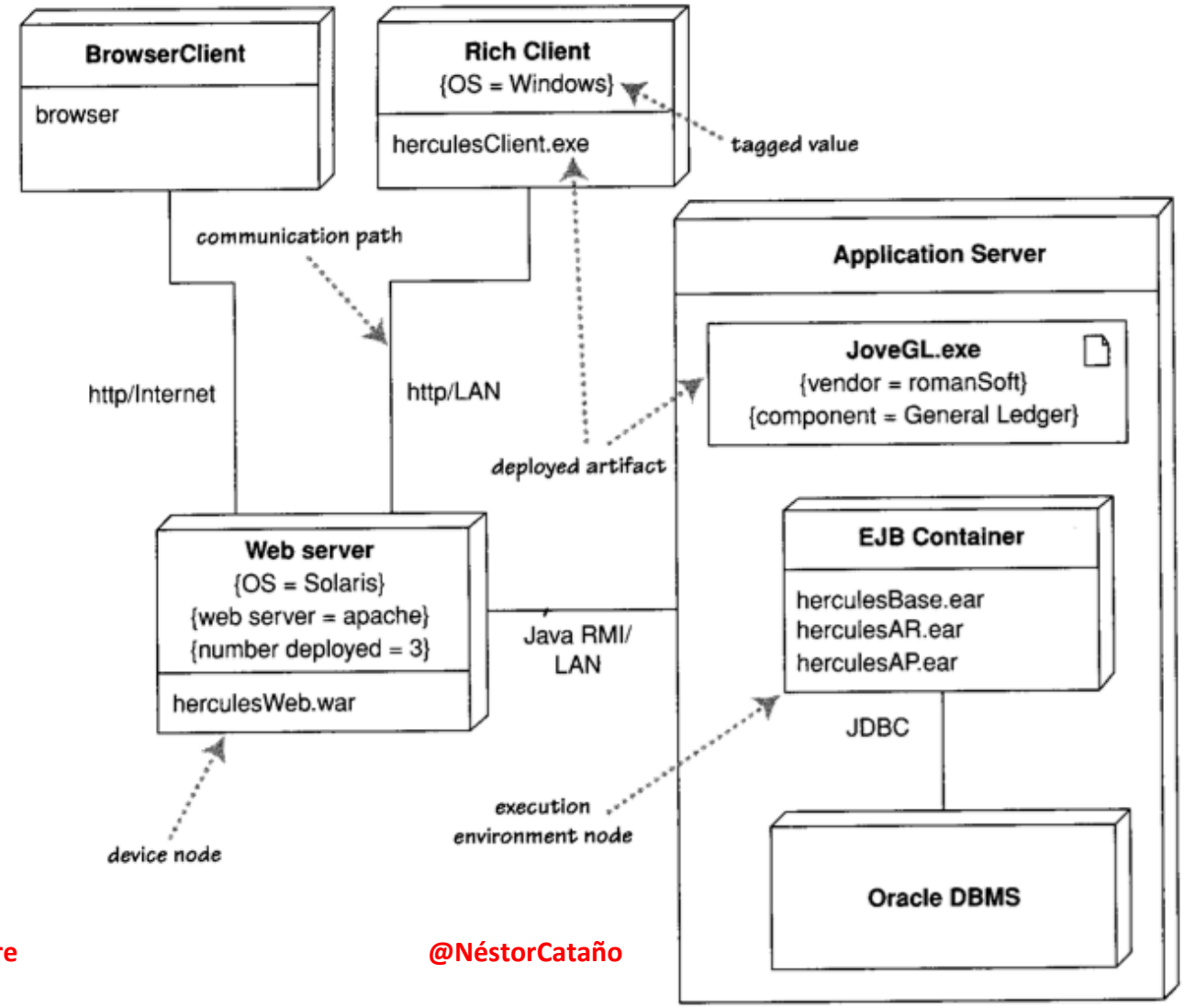
Useful Allocation Structures

- Deployment structure
- Implementation structure
- Work assignment structure

Allocation Structures: Deployment Structure

- **Deployment Structure**
  - Shows how **software** is assigned to **hardware** processing and communication elements.
  - Elements are software processes, processors, and communication pathways.
  - **Key Architectural Drivers** → **Performance**, **Security** (who can access what) and **Availability** (proportion of time a system is working)
  - **Key UML Diagrams** → **Deployment Diagrams:**
    - Specification level deployment diagram
    - **Instance level deployment diagram**
    - Network architecture of the system

An UML Deployment Diagram for a Web System



Allocation Structures: Implementation Structure

- **Implementation Structure**
  - Shows how **software elements** (**modules**) are mapped into **file structures** in the system's development, integration, or configuration control environments.
  - **Key Architectural Drivers** → **Efficiency**

Allocation Structures:
Work Assignment Structure

- **Work Assignment Structure**
  - Assigns responsibility for implementing and integrating the modules to **teams** who will carry it out.
  - **Key Architectural Drivers** → **Efficiency**

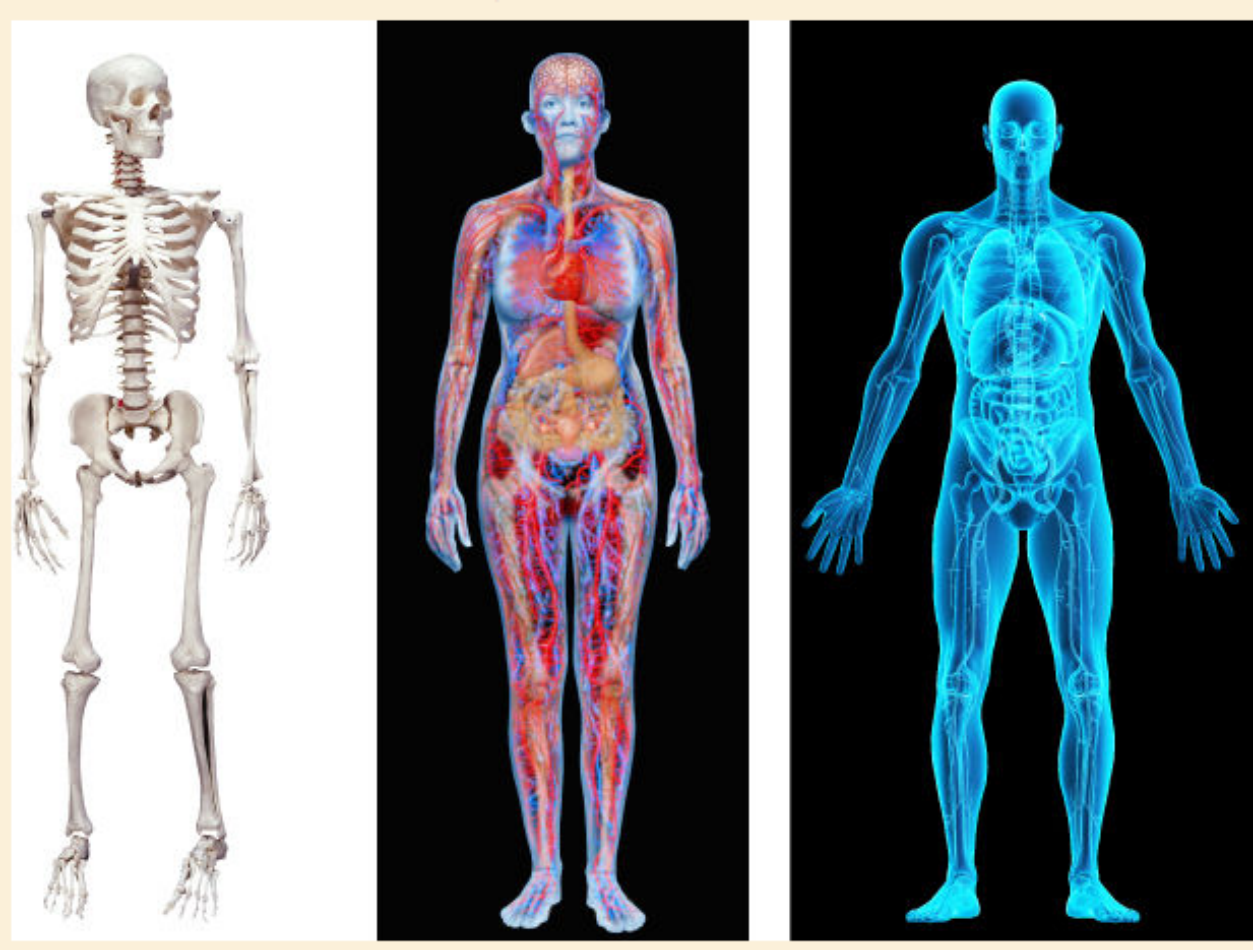
Useful Architectural Structures

	Software Structure	Element Types	Relations	Useful For	Quality Attributes Affected
Module Structures	Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control	Modifiability
	Uses	Module	Uses (i.e., requires the correct presence of)	Engineering subsets, engineering extensions	"Subsetability," extensibility
	Layers	Layer	Requires the correct presence of, uses the services of, provides abstraction to	Incremental development, implementing systems on top of "virtual machines"	Portability
	Class	Class, object	Is an instance of, shares access methods of	In object-oriented design systems, factoring out commonality; planning extensions of functionality	Modifiability, extensibility
	Data model	Data entity	{one, many}-to-{one, many}, generalizes, specializes	Engineering global data structures for consistency and performance	Modifiability, performance
C&C Structures	Service	Service, ESB, registry, others	Runs concurrently with, may run concurrently with, excludes, precedes, etc.	Scheduling analysis, performance analysis	Interoperability, modifiability
	Concurrency	Processes, threads	Can run in parallel	Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed	Performance, availability
Allocation Structures	Deployment	Components, hardware elements	Allocated to, migrates to	Performance, availability, security analysis	Performance, security, availability
	Implementation	Modules, file structure	Stored in	Configuration control, integration, test activities	Development efficiency
	Work assignment	Modules, organizational units	Assigned to	Project management, best use of expertise and available resources, management of commonality	Development efficiency

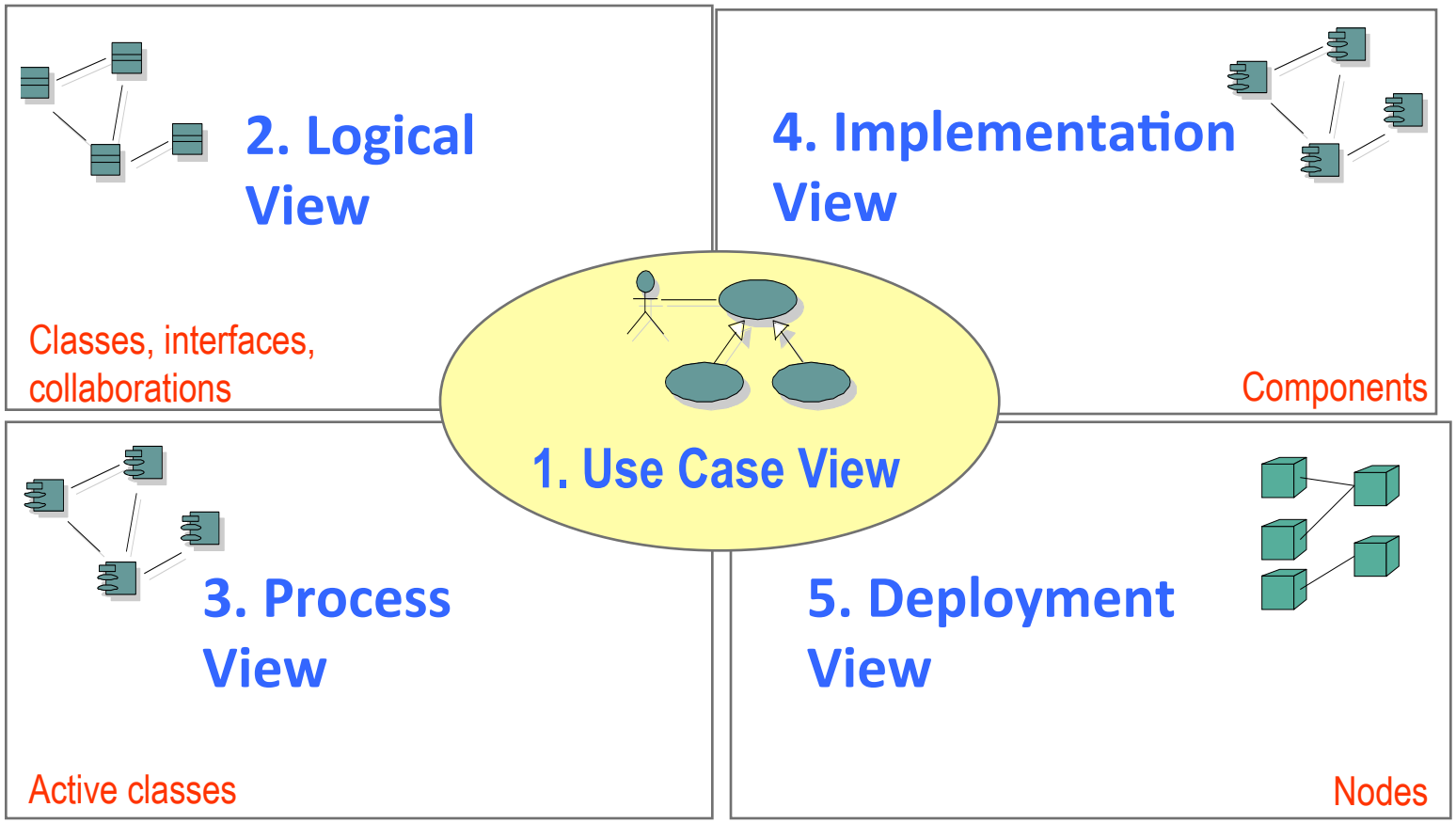
Architectural Structures and Views

- It is impossible to represent all information about a system's architecture in a single architectural model, as each model only shows one view or perspective of the system.
- **One usually needs to present several views of the software application**

Architectural Structures and Views



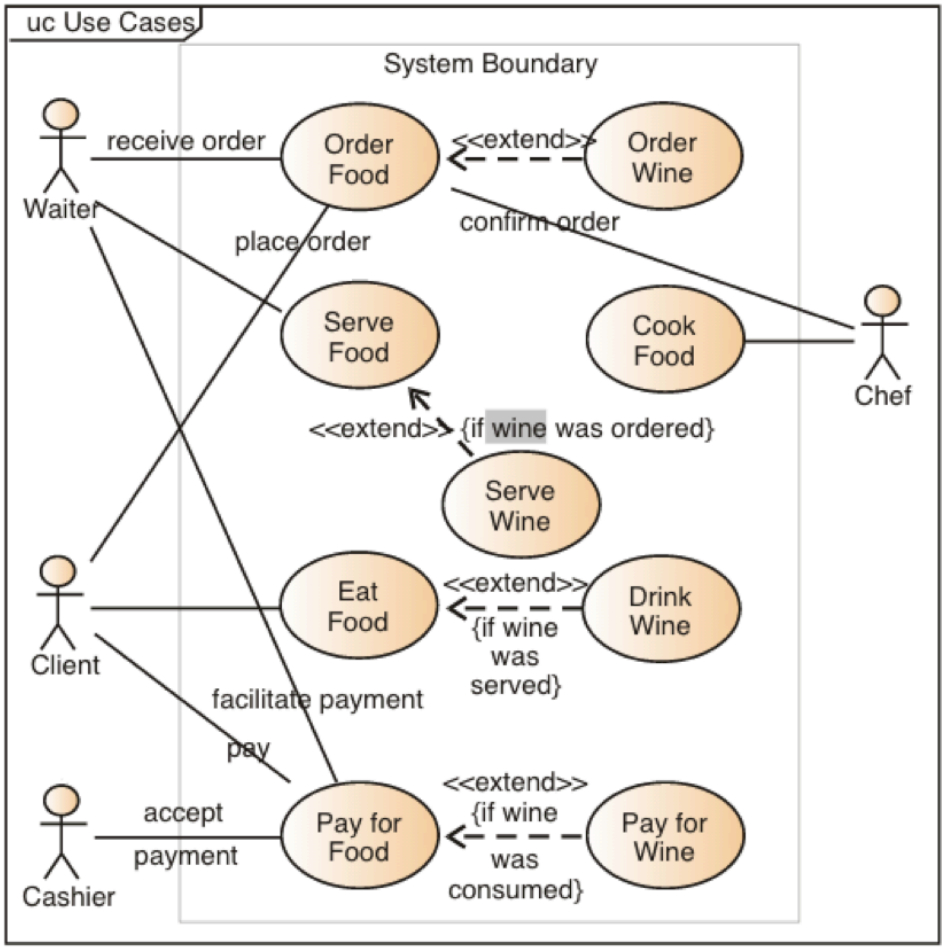
4+1 architectural view model



1. Use Cases

- Describes a typical interaction between a user and the system, providing a narrative on how the system is used
- Built in early stages of development
- A **scenario** is a sequence of steps describing an interaction between a user and a system
- **Actors** interact with the system

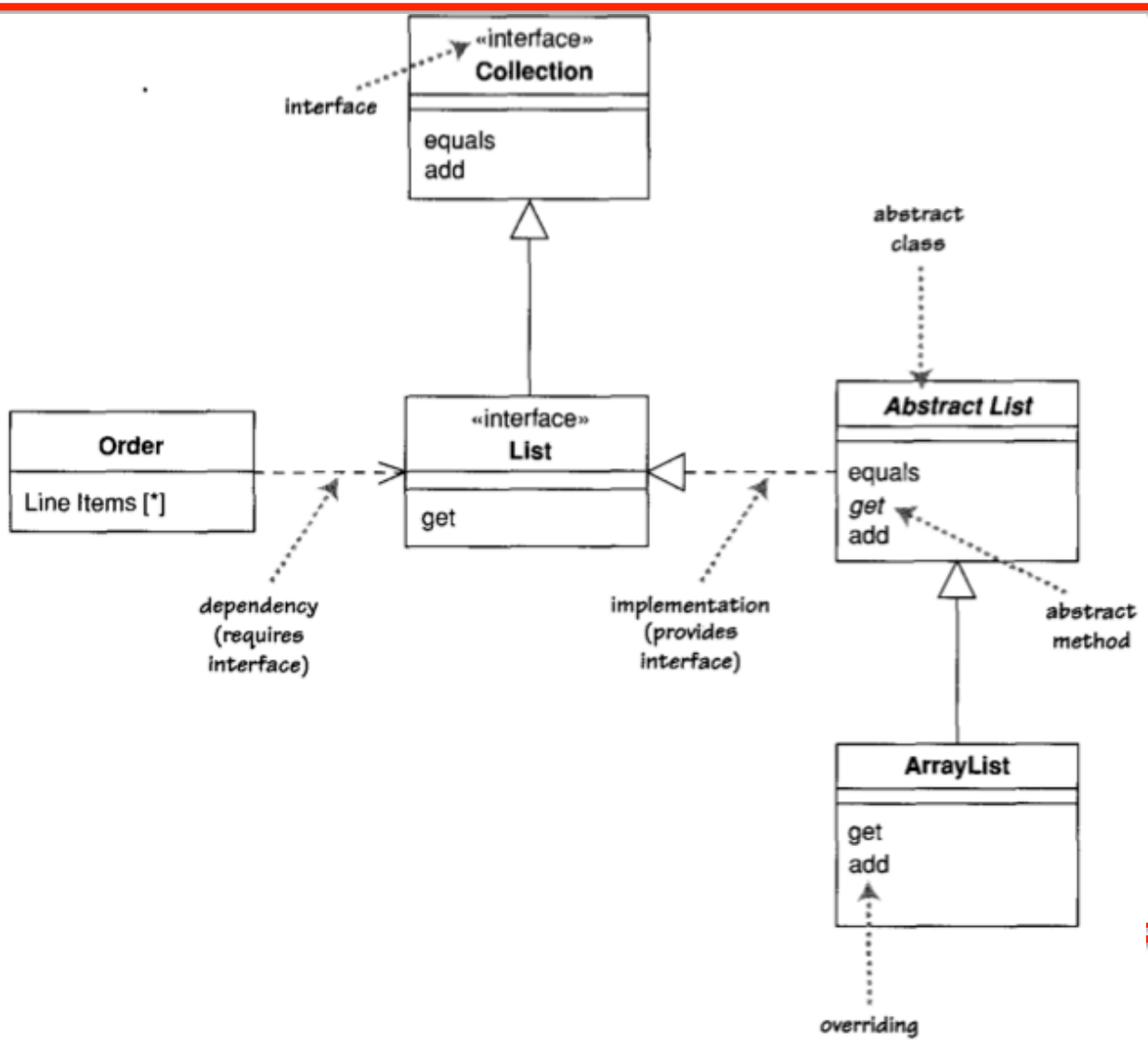
1. Use Cases


uc Use Cases
System Boundary
Waiter
receive order
Order Food
<<extend>>
Order Wine
confirm order
place order
Serve Food
Cook Food
Chef
<<extend>>
{if wine was ordered}
Serve Wine
Client
Eat Food
<<extend>>
{if wine was served}
Drink Wine
facilitate payment
pay
Cashier
accept payment
Pay for Food
<<extend>>
{if wine was consumed}
Pay for Wine

2. Logical View

- The purpose of the the **Logical View** is to capture the **functional requirements** of the system
- The **Logical View** shows key abstractions in the system as objects or classes.
- The **Logical View** relates system requirements to entities in the logical view.
- **Key UML Diagrams**
  - **Class Diagram**
  - **Sequence Diagram**
  - **Communication Diagram**

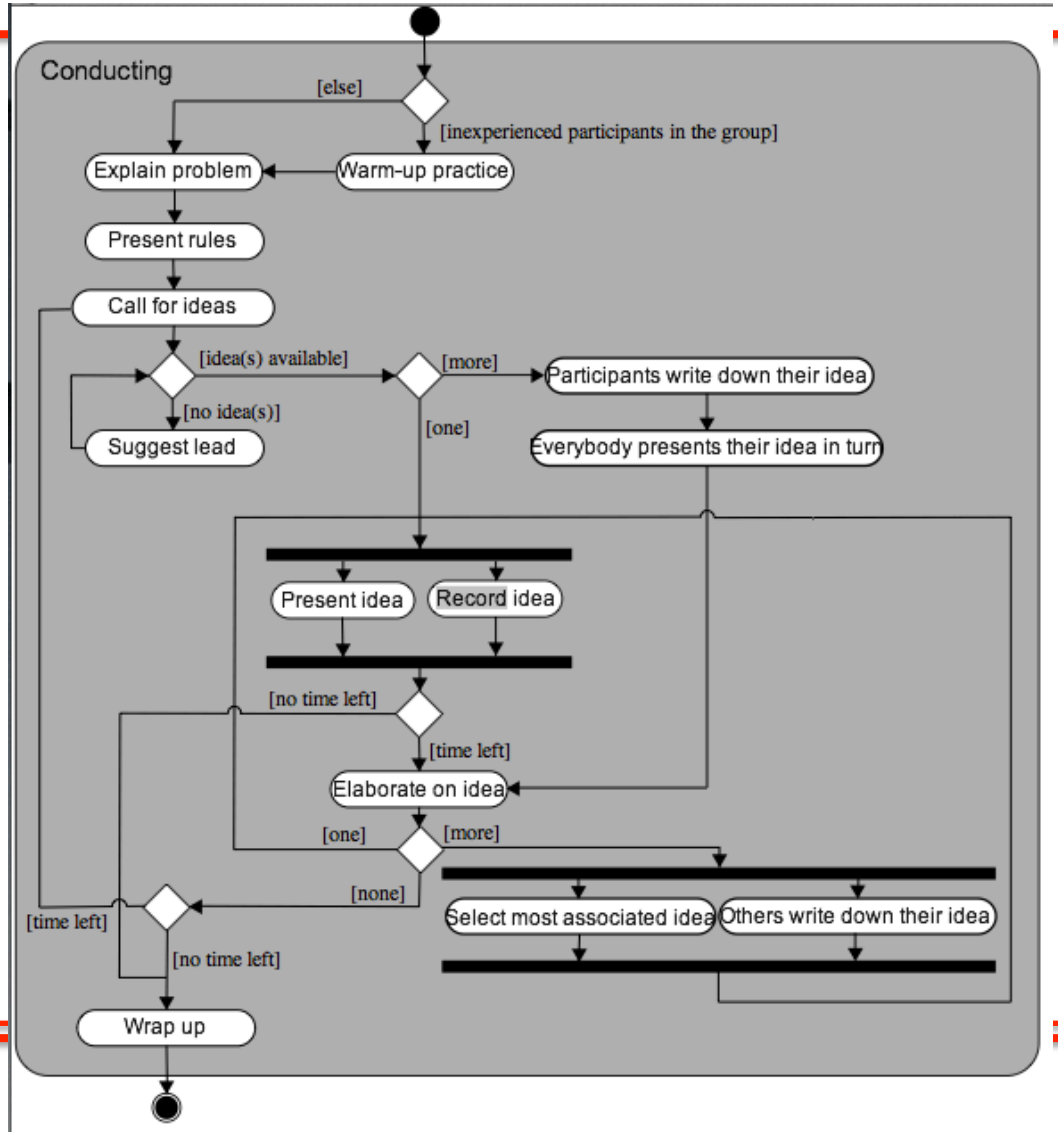
A Class Diagram for a Java Library



3. Process View

- Describes processes and threads that are part of the system concurrency and synchronization mechanisms
- Shows at runtime how the system is composed of **interacting processes**.
- **Key UML Diagrams**
  - **Activity Diagrams**
  - **State transition diagrams**

Activity diagram for brainstorming



4. Implementation View

- shows the break-down of the software into components that are to be implemented.
- How software is assigned to hardware processing and communicating elements?
- **Key UML Diagrams →**
  - **Composite Diagrams**
  - **Component Diagrams**

5. Deployment View

- shows how software components are distributed across the processors in the system
- **Key UML Diagrams**
  - **Deployment diagrams**

Further Reading

- **Books**
  - Software Engineering by Sommerville
  - UML Distilled by Fowler
  - Software Architecture in Practice by Kazman