POLITECNICO
MILANO 1863

Software Engineering 2

Introduction to Software Design

Software Architecture as a set of structures

Role of UML in Software Architecture

Exercise: reasoning on architectural documentation

M Camilli, E Di Nitto, M Rossi

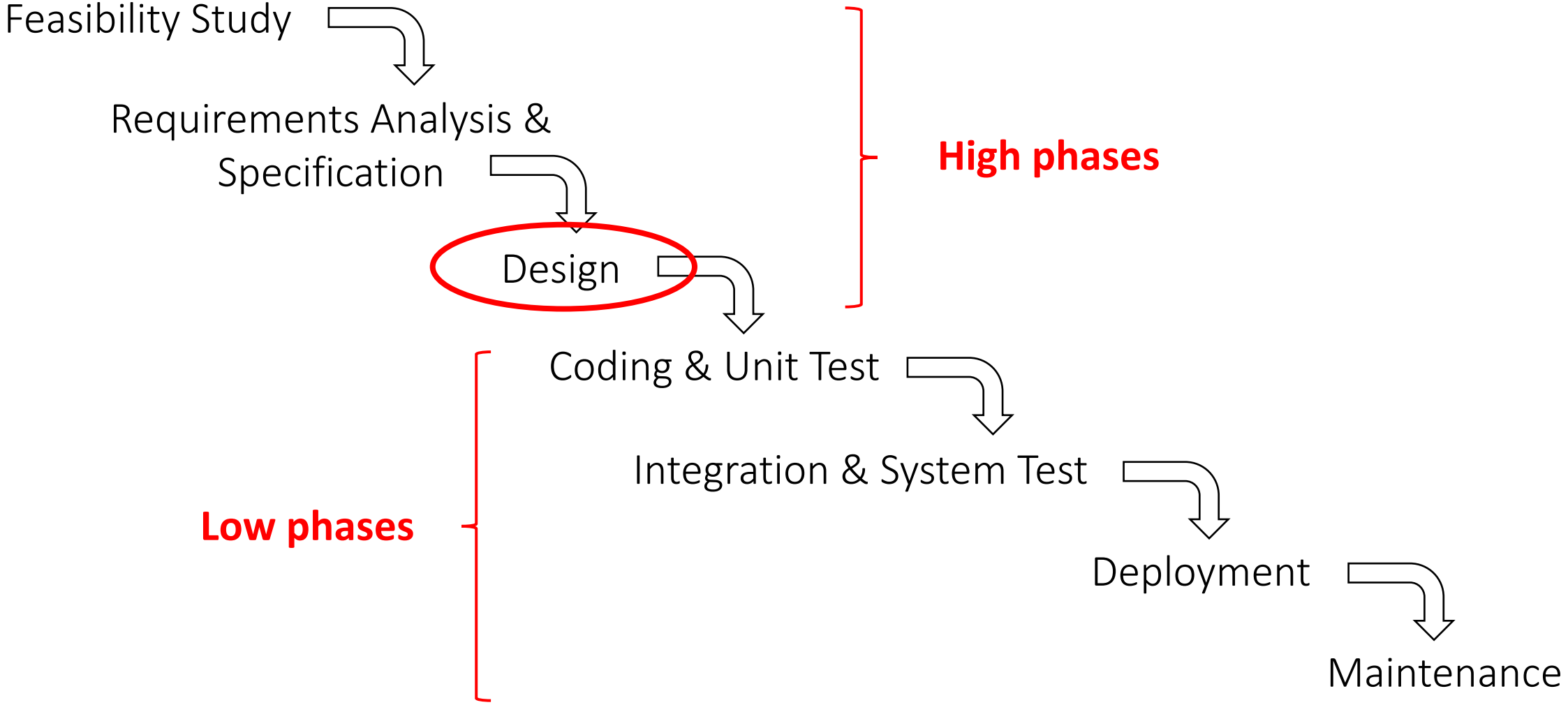
POLITECNICO
MILANO 1863

Software Design

Introduction

Structures and role of UML

A waterfall organization



Software design

- It’s all about making structural decisions
- Defines the software architecture
  - Components (modules)
  - Relations among components
  - Interactions among components
- Main goals
  - Support trade-off analysis (through **reasoning-enabling structures**)
  - Enable concurrent development
  - Separate responsibilities
  - Increase understandability of the solution
- Produces the **Design Document (DD)**

What is a Software Architecture?

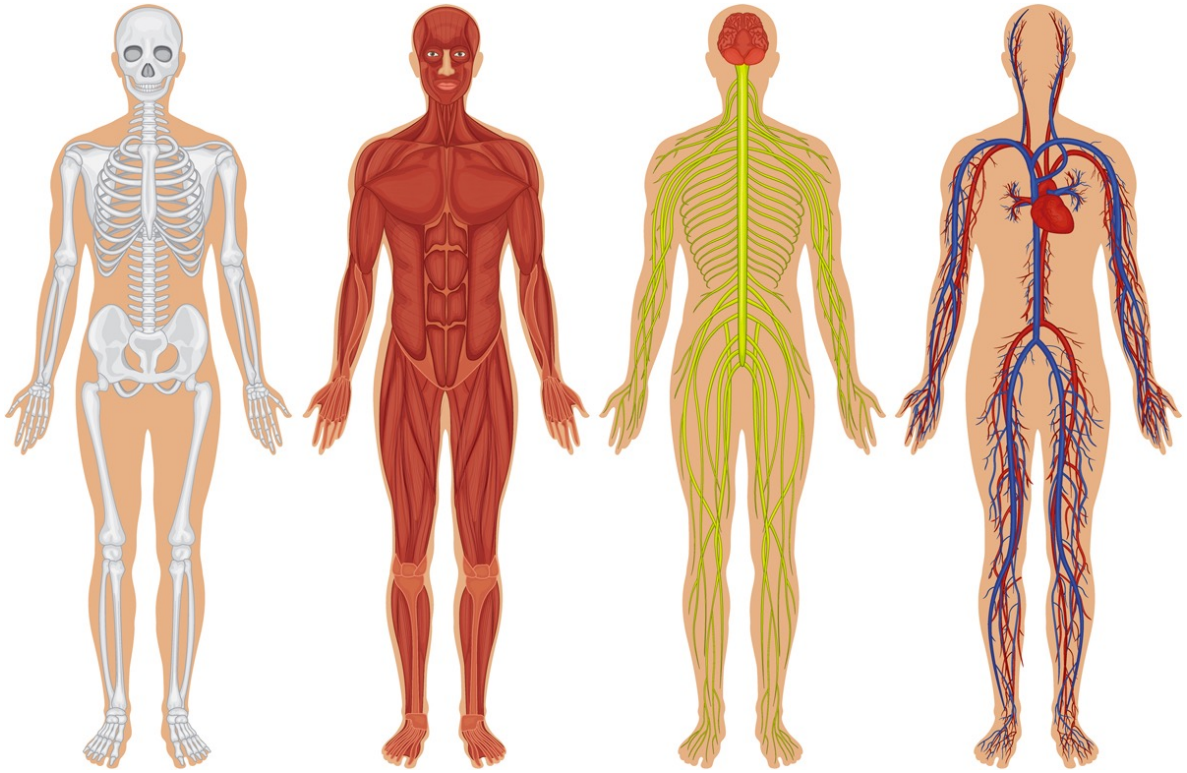
POLITECNICO MILANO 1863

- The Software Architecture (SA) of a system is the set of structures needed to reason about the system. These structures comprise software elements, relations among them, and properties of both. [Bass et al. 2021]
- SA as a tool to reason about systems
- SA composed of a set of structures

[Bass et al. 2021] *Bass, Len, et al. Software Architecture in Practice, Pearson Education, Limited, 2021.*

Set of structures, counterparts in nature

- Human body example
  - Different views of **various structures** of a human body
  - All views have **very different properties**
  - All views are **inherently related** and interconnected
  - Toghether they describe the “architecture” of the human body



Set of structures relevant to software

POLITECNICO MILANO 1863

- Component-and-connector structures
- Module structures
- Allocation structures

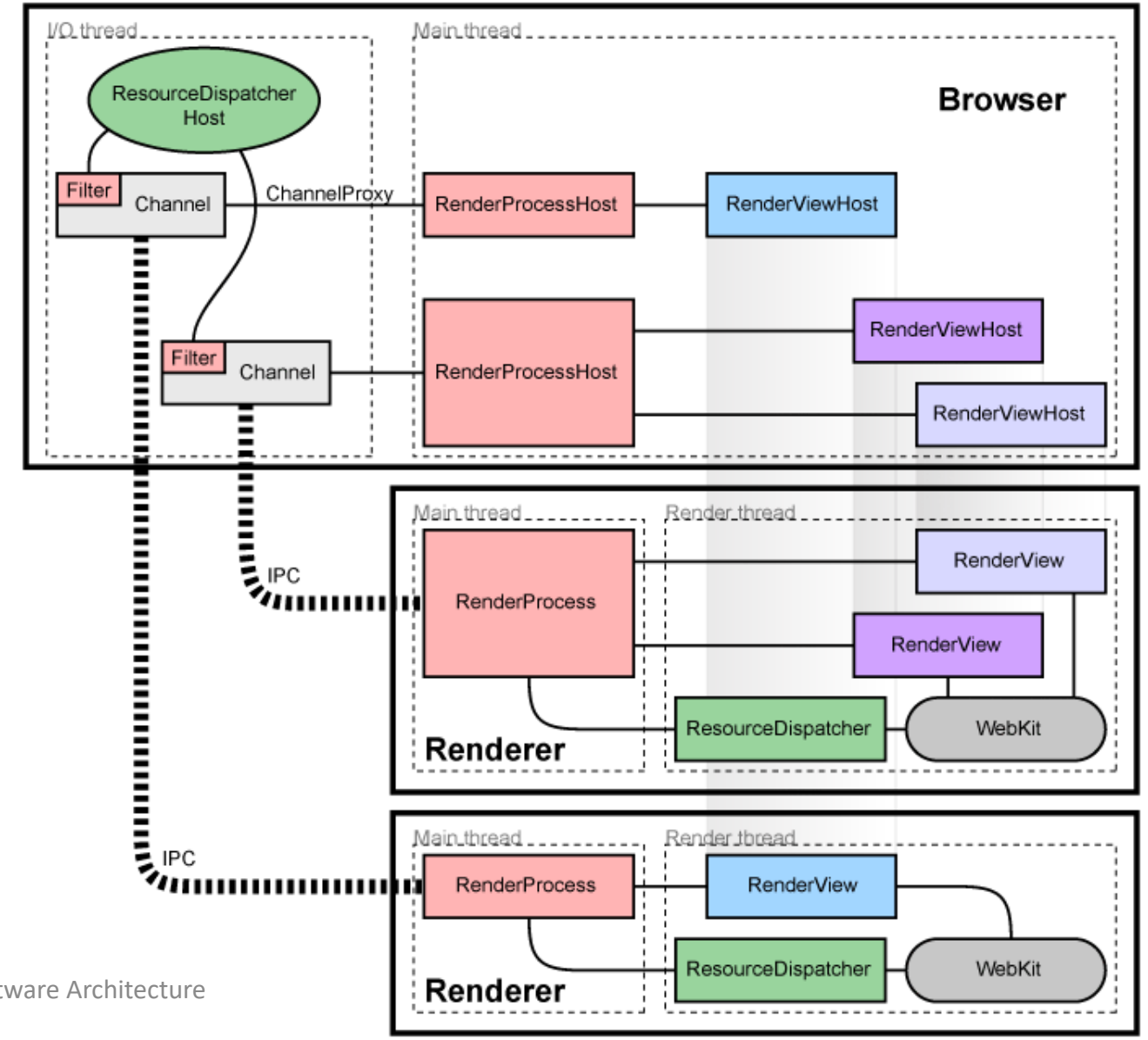
Component-and-connector structures

POLITECNICO MILANO 1863

- Describe how the system is structured as a set of elements that have runtime behavior (components) and interactions (connectors).
- **Components**
  - Principal units of computation
  - Examples: clients, servers, services, …
- **Connectors**
  - Communication means
  - Examples: request-response mechanisms, pipes, asynchronous messages, …

An example of component-and-connector structure

- From the Chromium documentation
  https://www.chromium.org/developers/design-documents/multi-process-architecture

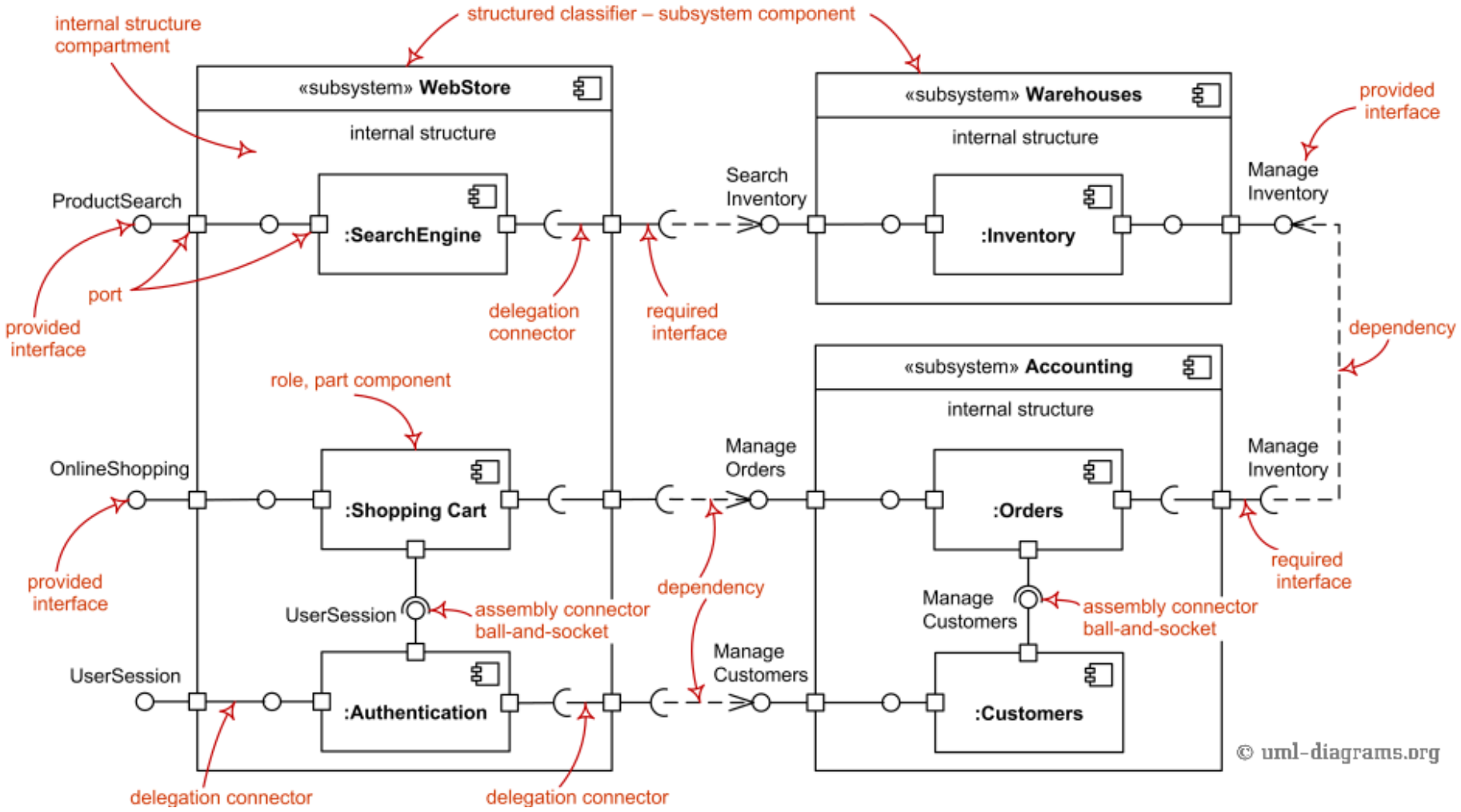


Component-and-connector structures: purpose

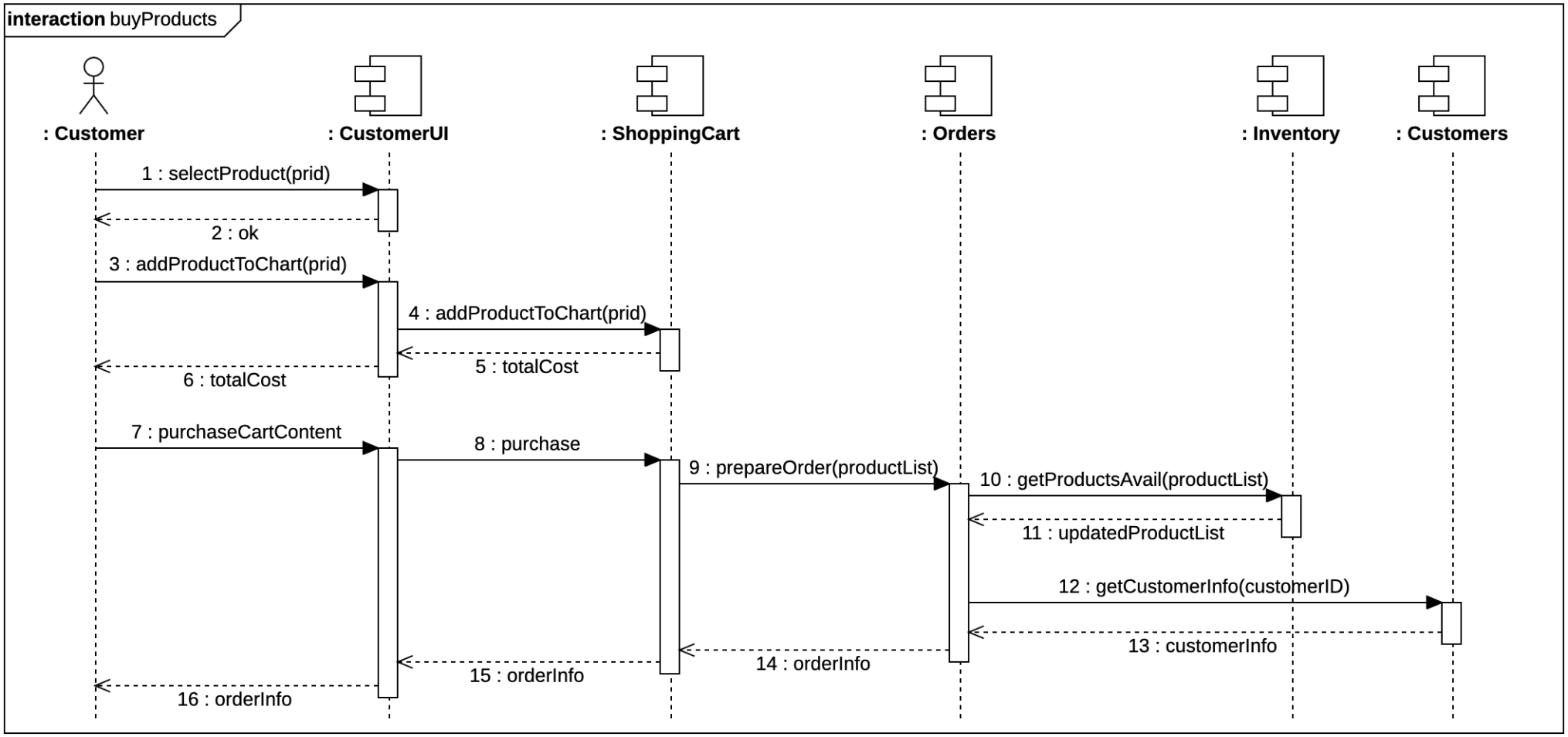
POLITECNICO MILANO 1863

- Allow us to answer to questions such as
  - 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?
  - Which parts of the system can run in parallel?
  - How does the system's structure evolve during execution?
- Also, allow us to study runtime properties such as availability and performance

Using UML component diagrams to represent component-and-connector structures



Using UML sequence diagrams to represent component-and-connector structures



Module structures

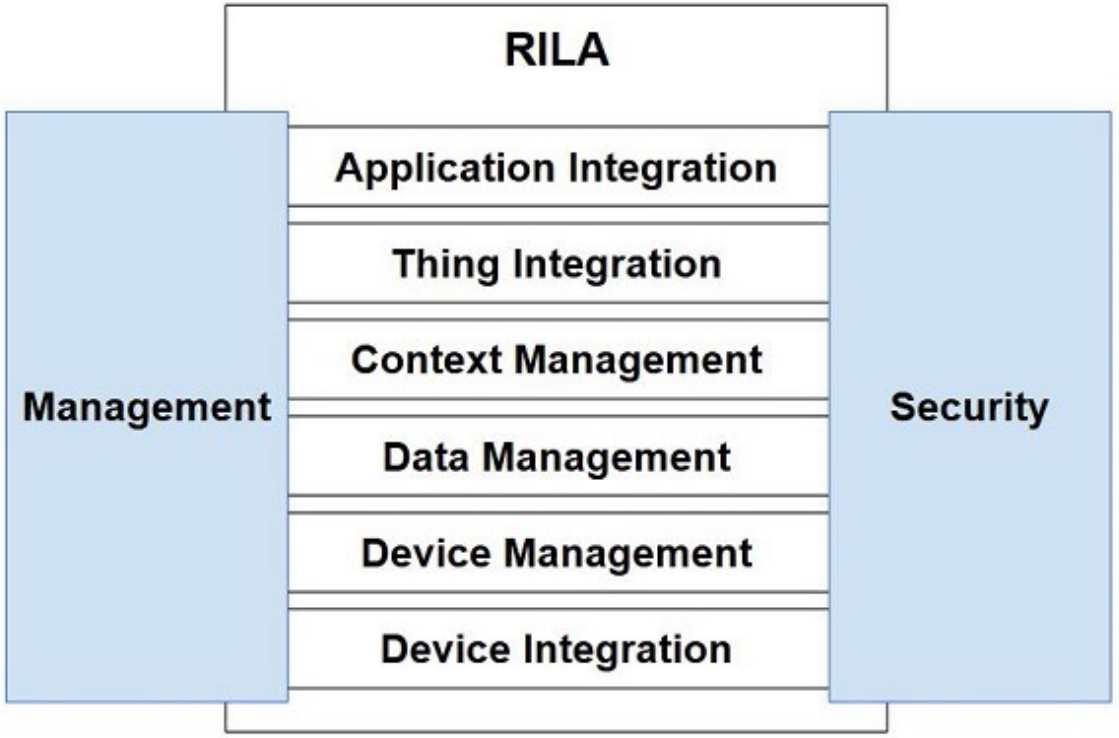
POLITECNICO MILANO 1863

- Show how a system is structured as a set of code or data units that have to be procured or constructed, together with their relations
- Examples of modules: packages, classes, functions, libraries, layers, database tables…
- Modules constitute implementation units that can be used as the basis for work splitting (identifying functional areas of responsibility)
- Typical relations among modules:
  - uses, is-a (generalization), is-part-of

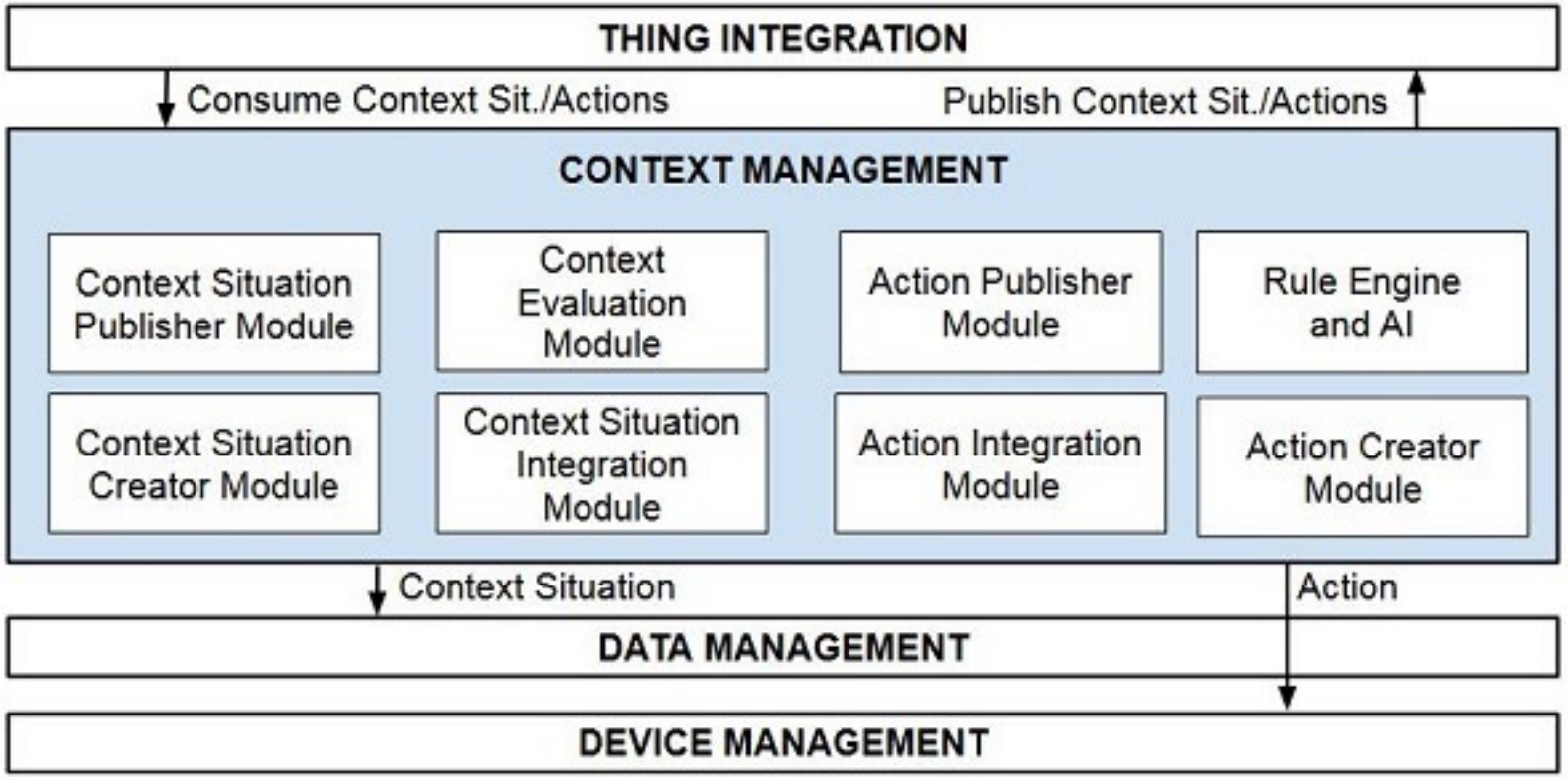
Example of modular structure: Reference IoT Layered architecture (RILA)

https://www.infoq.com/articles/internet-of-things-reference-architecture

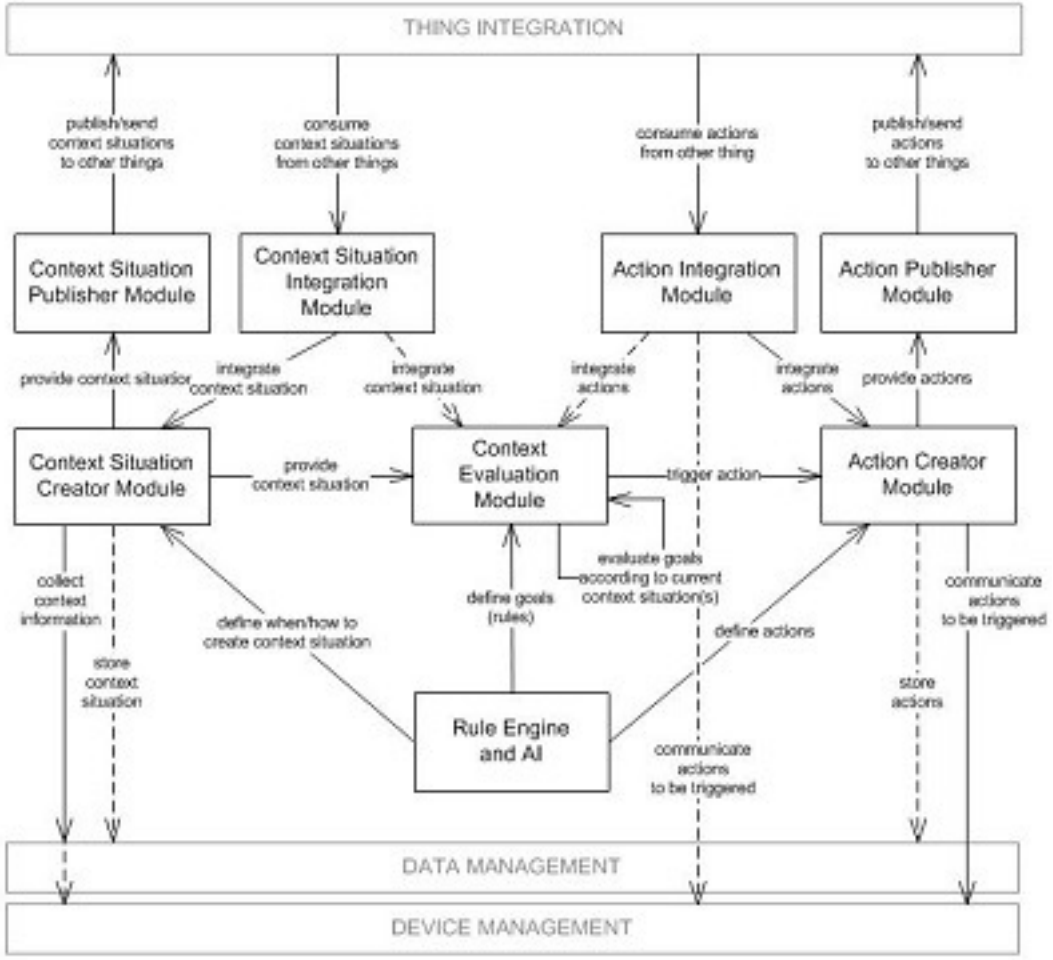
- **Layered architecture**:
  - Layers are organized according to use relationships
  - A layer can use the layer below and can be used by the layer above



RILA (cont.)


THING INTEGRATION
Consume Context Sit./Actions
Publish Context Sit./Actions
CONTEXT MANAGEMENT
Context Situation Publisher Module
Context Evaluation Module
Action Publisher Module
Rule Engine and AI
Context Situation Creator Module
Context Situation Integration Module
Action Integration Module
Action Creator Module
Context Situation
Action
DATA MANAGEMENT
DEVICE MANAGEMENT

RILA (cont.)


THING INTEGRATION
publish/send context situations to other things
consume context situations from other things
consume actions from other thing
publish/send actions to other things
Context Situation Publisher Module
Context Situation Integration Module
Action Integration Module
Action Publisher Module
provide context situation
integrate context situation
integrate context situation
integrate actions
integrate actions
provide actions
Context Situation Creator Module
provide context situation
Context Evaluation Module
trigger action
Action Creator Module
collect context information
store context situation
define when/how to create context situation
define goals (rules)
evaluate goals according to current context situation(s)
define actions
communicate actions to be triggered
store actions
Rule Engine and AI
communicate actions to be triggered
DATA MANAGEMENT
DEVICE MANAGEMENT

What can we understand from a layered organization?

POLITECNICO MILANO 1863

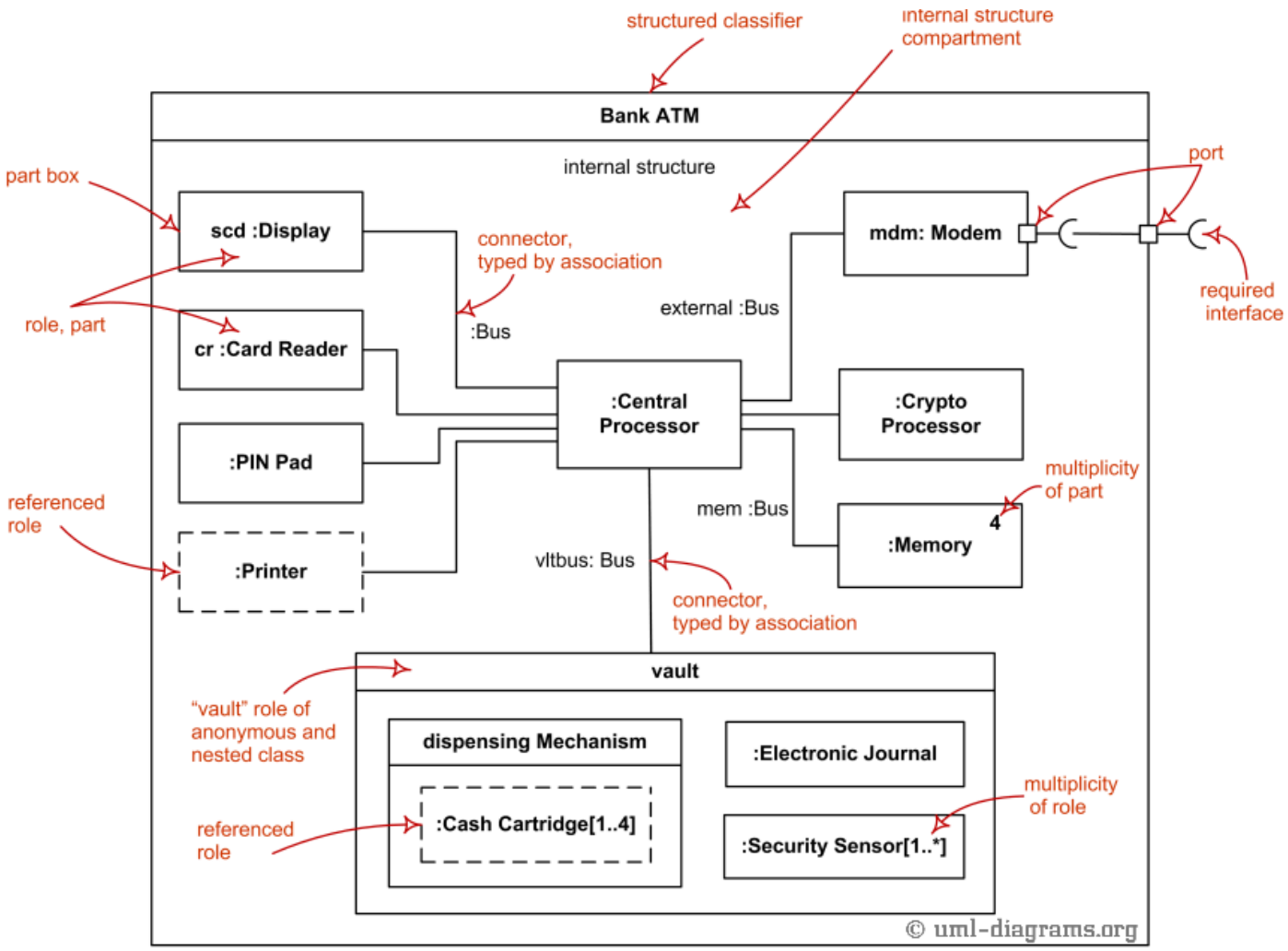
- Layers enable separation of concerns
- We can identify the main focus of each layer at a glance
- Each layer can be implemented by a different team

UML diagrams to represent modular structures

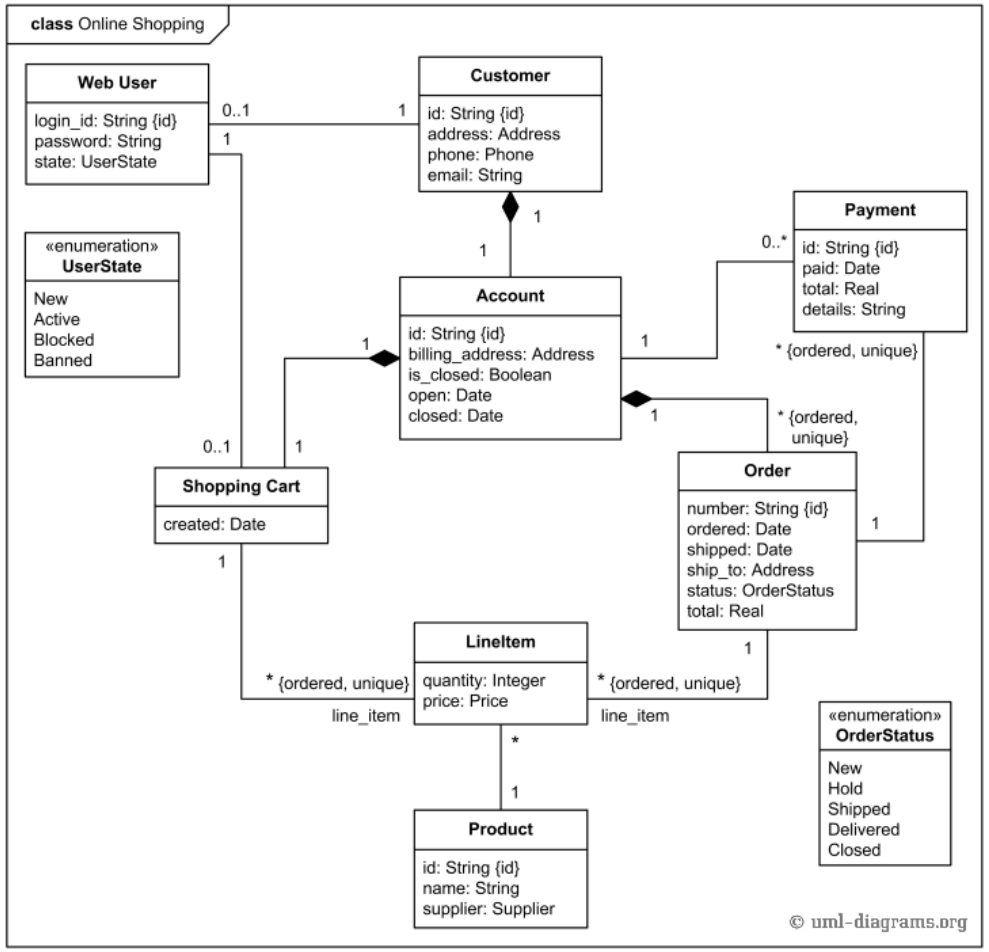
POLITECNICO MILANO 1863

- Composite structure diagram
- Class diagram
- Package diagram

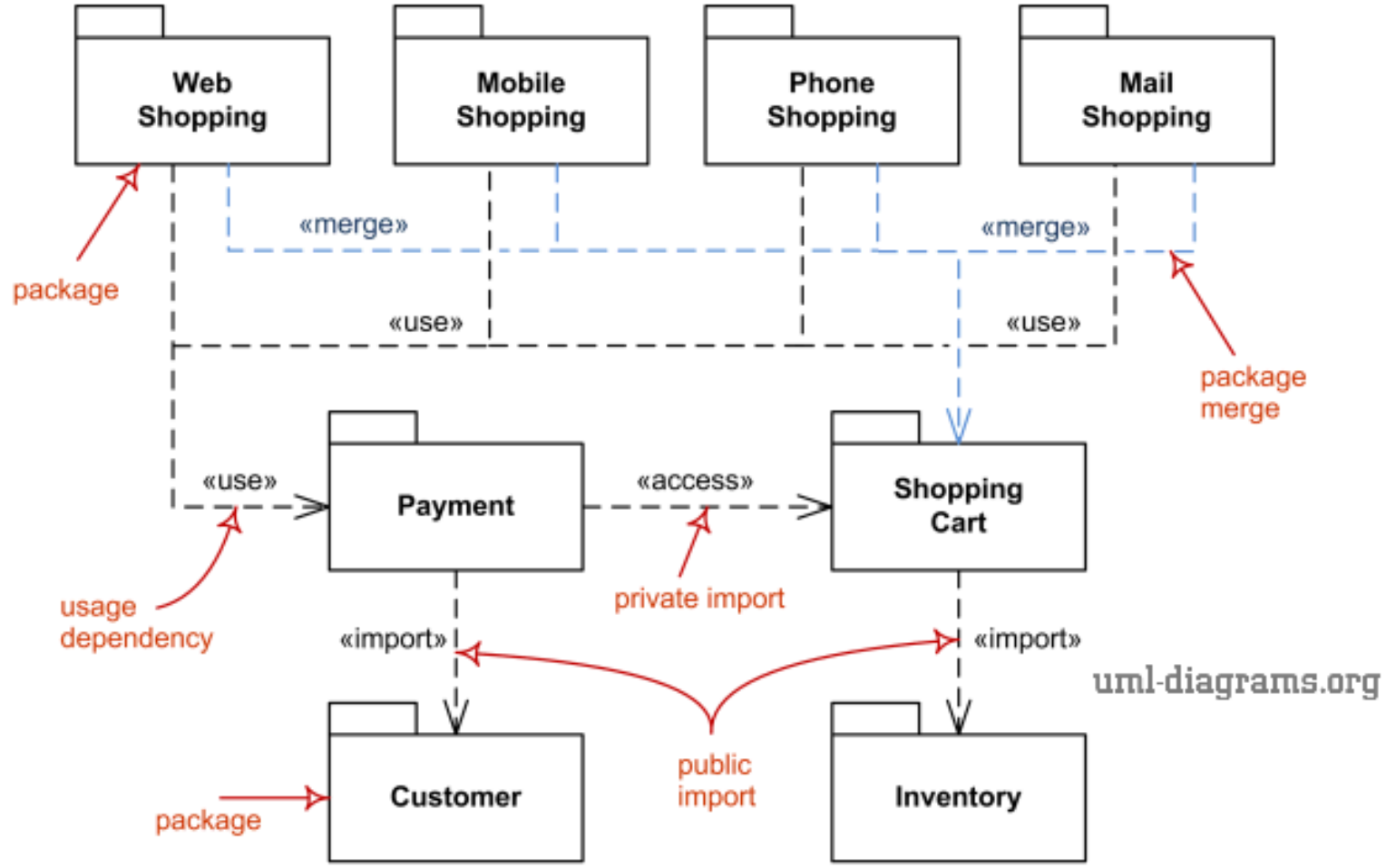
UML composite structure example



UML class diagram example



UML package diagram



Module structures: purpose

POLITECNICO MILANO 1863

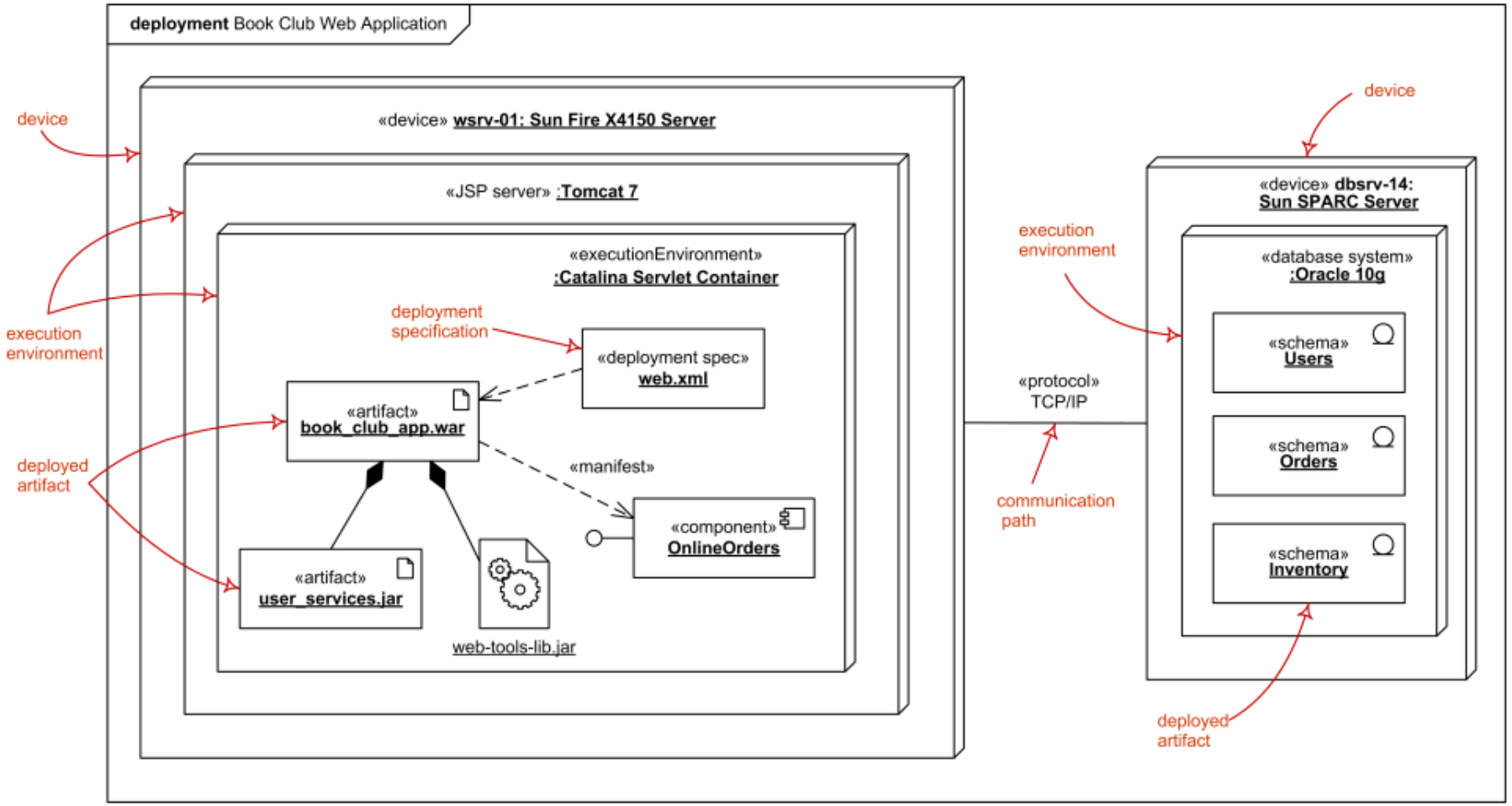
- Allow us to answer questions such as the following:
  - What is the primary functional responsibility assigned to each module?
  - What other software elements is a module allowed to use?
  - What other software does it actually use and depend on?
  - What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Can also be used to answer questions about the impact on the system when the responsibilities assigned to each module change
  - module structures are primary tools for reasoning on system's modifiability

Allocation structures

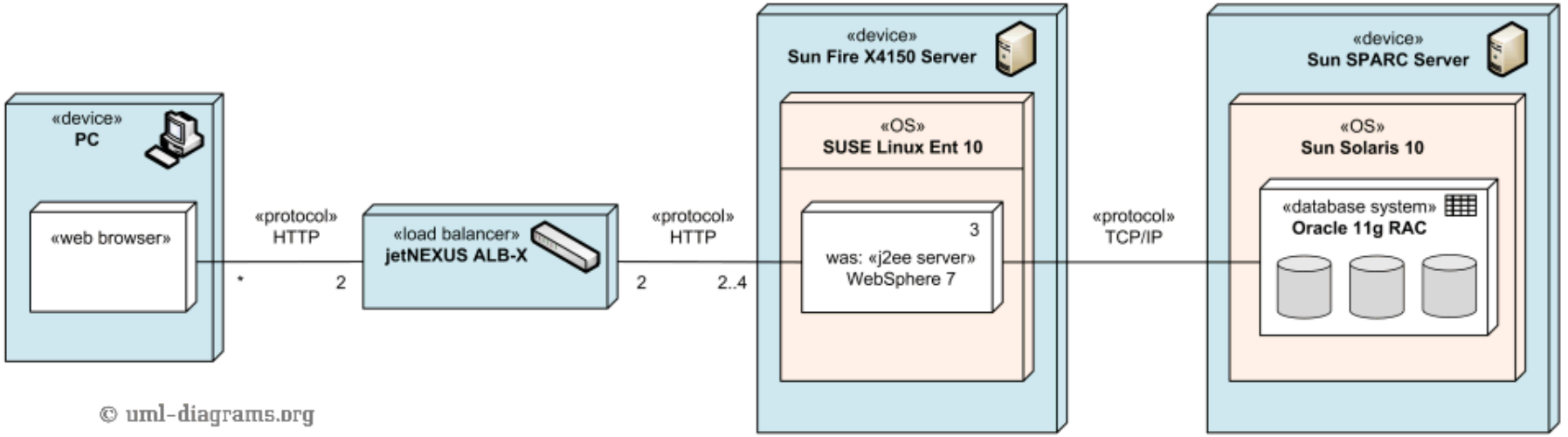
POLITECNICO MILANO 1863

- Define how the elements from C&C or module structures map onto things that are not software
- Examples of such things:
  - hardware (possibly virtualized),
  - file systems,
  - teams
- Typical allocation structures:
  - Deployment structure
  - Implementation structure
  - Work assignment structure

UML deployment diagrams and deployment structure



UML deployment diagrams and deployment structure



Deployment Structure

POLITECNICO MILANO 1863

- Captures the topology of a system's hardware
- Built as part of architectural specification
- Purpose:
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

Summary of structures and their purposes

POLITECNICO MILANO 1863

Classes of structures	Structures	Element types	Relations	Useful for
Modular structures	Composite structures, package diagrams	Modules, packages	Is a submodule of, uses	Resource allocation, project planning, encapsulation
	Class diagram	Classes	Is-a, is part of	Object-oriented development, planning for extensions
	Layered structures		Can use, provides abstractions to	Incremental development
	Data model	Data entities	One-to-one, one-to-many, many-to-one, many-to-many, is-a	Engineering global data structures for consistency and performance

Summary of structures and their purposes

POLITECNICO MILANO 1863

Classes of structures	Structures	Element types	Relations	Useful for
Component& Connector structures	Component diagrams	Components offering services	Provided and required interfaces	Performance analysis, robustness analysis, resource allocation, project planning
	Sequence diagrams	Processes/threads	Synchronous and asynchronous messages	Analysis of resource contention, parallelism opportunities
Allocation structures	Deployment diagrams	Components, hardware/software execution environments	Allocated to	Performance, security, roboustness analyses
	Implementation structures	Modules, file structures	Stored in	Configuration control, integration, test activites
	Work assignment	Modules, organizational units	Assigned to	Project management, development efficiency

POLITECNICO
MILANO 1863

Software Design

The architecture is intrinsic to our software!

A simple example: a trivial prototype of a “banking application”

Banking application – Account

Code is available here: github.com/matteocamilli/bank_example_rmi

```
import java.rmi.*;
public interface Account extends Remote {
    public float balance() throws RemoteException;
}

import [...]
public class AccountImpl extends UnicastRemoteObject
                         implements Account {
    // implementation of class (method balance, etc.)
}
```

Banking application – AccountFactory

Code is available here: github.com/matteocamilli/bank_example_rmi

```
import java.rmi.*;
public interface AccountFactory extends Remote {
    public Account createAccount(String userName, int balance)
                    throws RemoteException;
    public int getLoad() throws RemoteException;
}

import [...]
public class AccountFactoryImpl extends UnicastRemoteObject
                                implements AccountFactory {
    // implementation of class (methds createAccount, getLoad, etc.)
    // …
}
```

Banking application – AccountManager

Code is available here: github.com/matteocamilli/bank_example_rmi

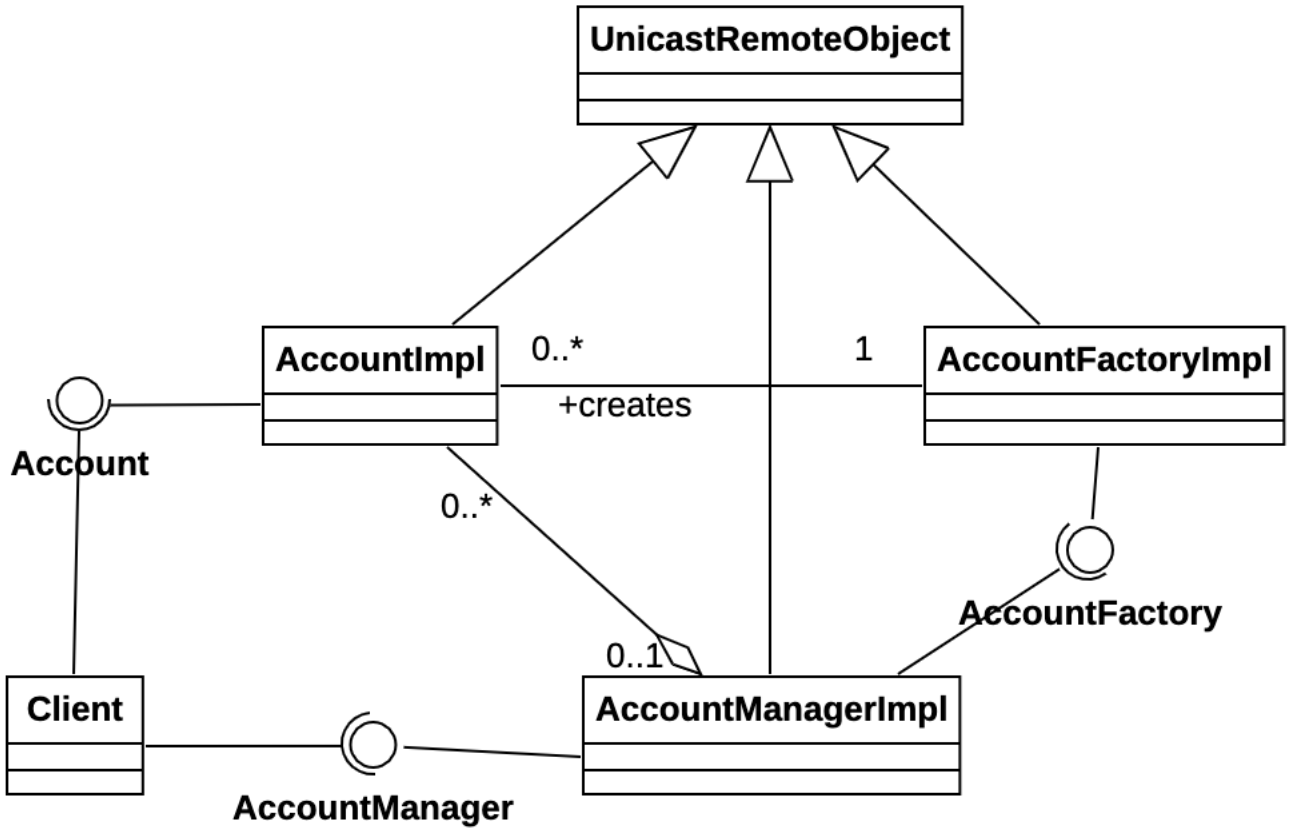
```
import java.rmi.*;
public interface AccountManager extends Remote {
        public Account open(String name) throws RemoteException;
}

import [...]
public class AccountManagerImpl extends UnicastRemoteObject
                                implements AccountManager {

    private Map<String,AccountFactory> accounts = new HashMap< >();
    // implementation of class (attributes, method open, etc.)
    public void addFactory(String hostName) { … }
    public AccountFactory selectFactory() { … }
}
```

Banking application – Client

Code is available here: github.com/matteocamilli/bank_example_rmi

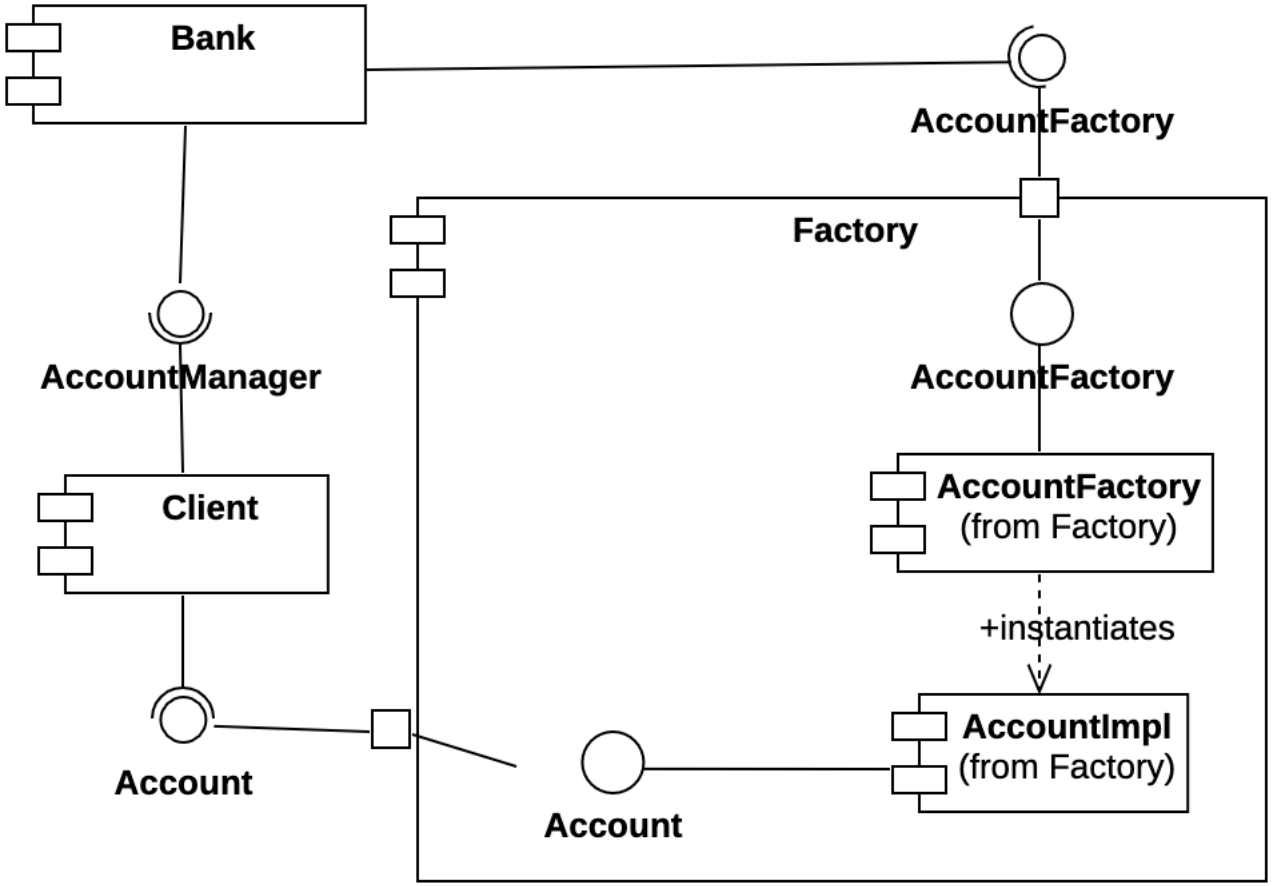
```
import java.rmi.*;
public class Client {
    public static void main (String[] args) {
        try {
            // … some stuff
            AccountManager manager =
                (AccountManager) Naming.lookup( "//" + args[0] + "/AccountManager" );
            String name = "Jack B. Quick";
            Account account = manager.open(name);
            float balance = account.balance();
            System.out.println ("The balance in " + name + "'s account is $" + balance);
        } catch (Exception e) {  e.printStackTrace(); }
    }
}
```

Banking application – Class diagram

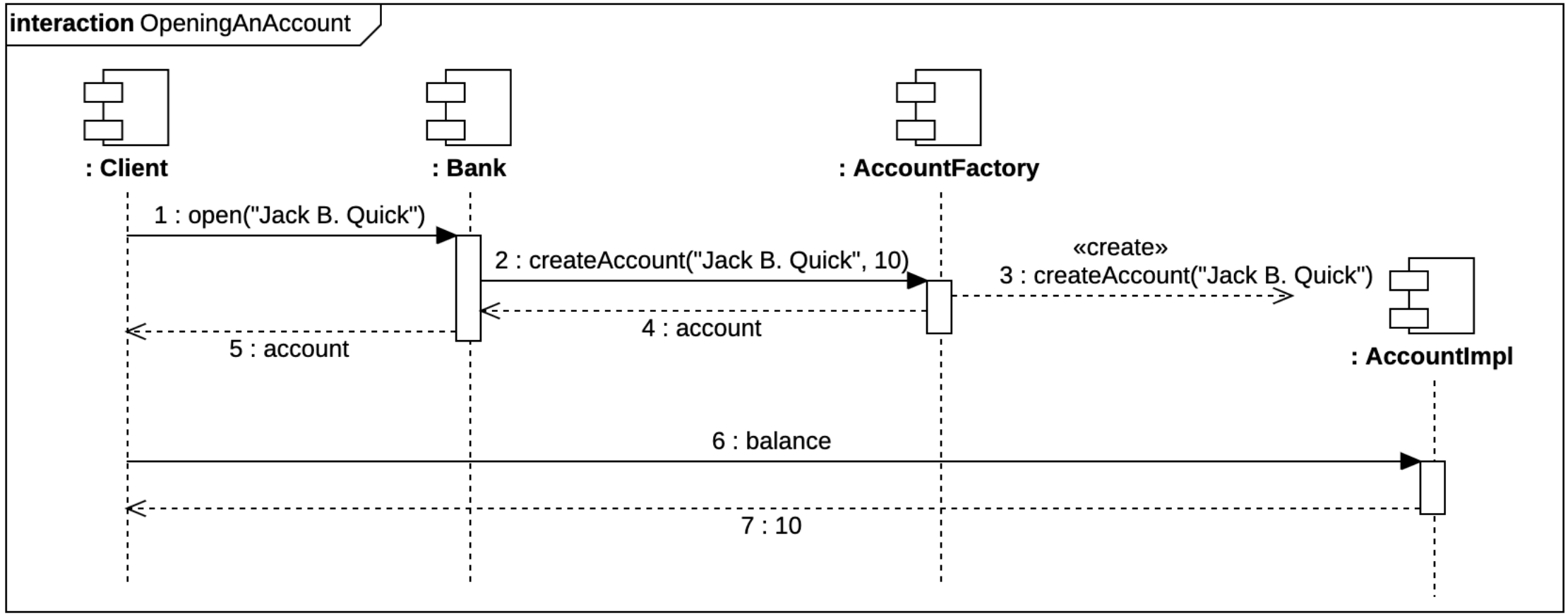


- What info do I get from this?
  - Hierarchy
  - The client uses remote objects (sends messages to objects that extend `UnicastRemoteObject`)
  - Client uses two interfaces
  - The other interface is used by `AccountManager`

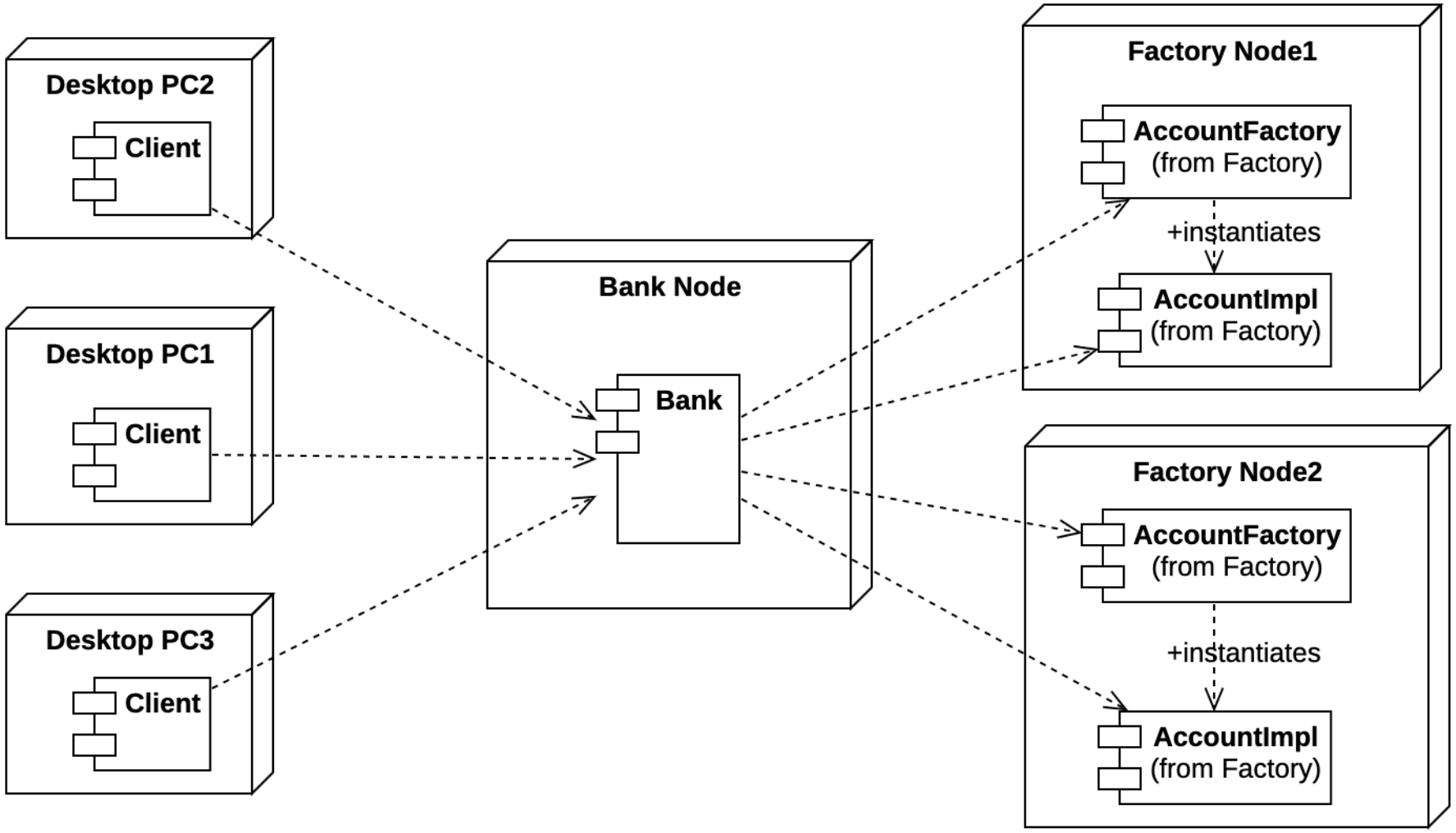
Banking application – Component diagram



Banking application – Sequence diagram



Banking application – Deployment diagram



POLITECNICO
MILANO 1863

Software Design

Exercise: Reasoning on architectural documentation — WE2 exam of Feb 10, 2023

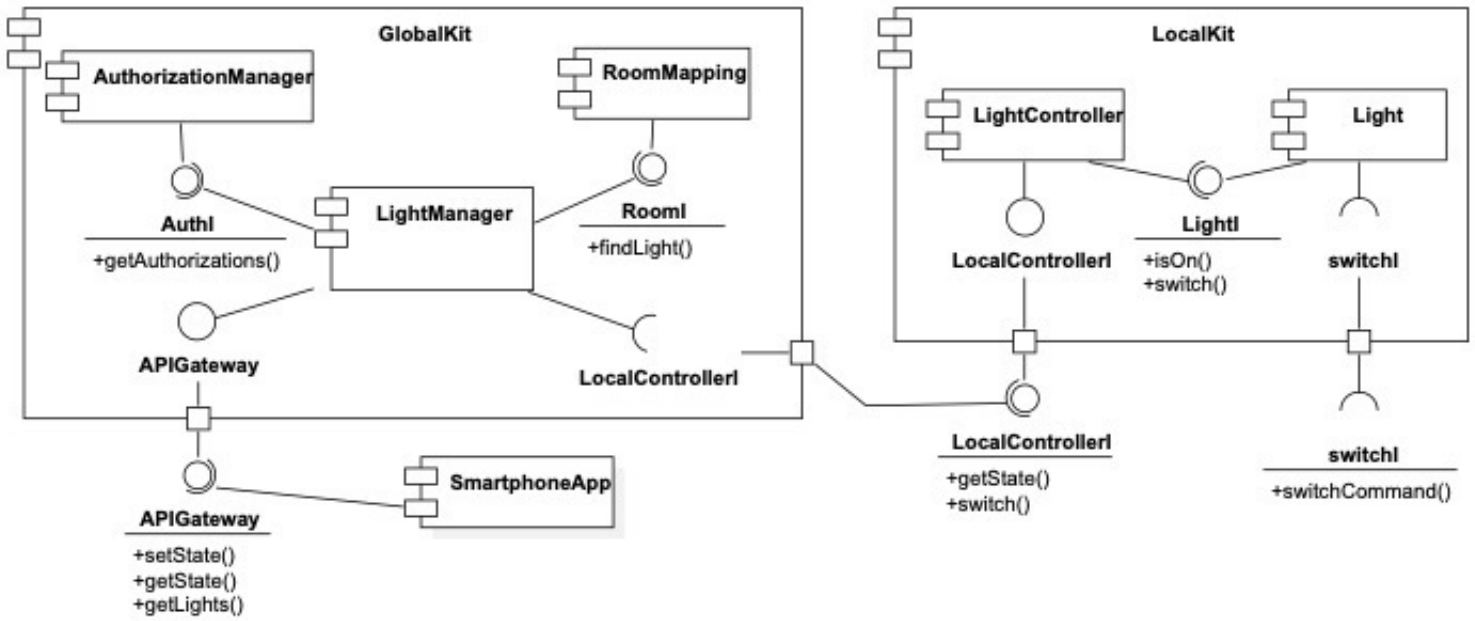
SmartLightingKit

POLITECNICO MILANO 1863

- SmartLightingKit is a system expected to manage the lights of a potentially big building composed of many rooms (e.g., an office space).
- Each room of the building, may have one or more lights.
- The system shall allow the lights to be controlled – either locally or remotely – by authorized users. Local control is achieved through terminals installed in the rooms (one terminal per room). Remote control is realized through a smartphone application, or through a central terminal installed in the control room of the building.
- While controlling the lights of a room, the user can execute one or more of the following actions: turn a light on/off at the current time, at a specified time, or when certain events happen (e.g., a person enters the building or a specific room). Moreover, users can create routines, that is, scripts containing a set of actions.
- SmartLightingKit manages remote control by adopting a fine-grained authorization mechanism. This means there are multiple levels of permissions that may even change dynamically.
- System administrators can control the lights of every room, install new lights, and remove existing ones. Administrators can also change the authorizations assigned to regular users. These last ones can control the lights of specific rooms. When an administrator installs a new light in a room, he/she triggers a pairing process between the light and the terminal located in that room. After pairing, the light can be managed by the system.

SmartLightingKit – part 1

- This diagram describes the portion of the SmartLightingKit system supporting lights turning on/off and lights status checking.



- What can we infer from the analysis of this diagram?

SmartLightingKit – part 2

POLITECNICO MILANO 1863

- The diagram is complemented by the following description
  - `GlobalKit` is the component installed in the central terminal. It can be contacted through the `APIGateway` interface by the `SmartphoneApp` component representing the application used for remote control. `GlobalKit` includes:
    - `RoomMapping`, which keeps track, in a persistent way, of lights' locations within the building's rooms;
    - `AuthorizationManager`, which keeps track, in a persistent way, of the authorizations associated with each user (for simplicity, we assume that users exploiting the operations offered by the `APIGateway` are already authenticated through an external system and include in their operation calls a proper token that identifies them univocally); and
    - `LightManager`, which coordinates the interaction with all `LocalKit` components.
  - Each `LocalKit` component runs on top of a local terminal. Also, each `LocalKit` exposes the `LocalControllerI` interface that is implemented by its internal component `LightController`, which controls the lights in the room. Each light is represented in the system by a `Light` component which interacts with the external system operating the light through the `switchCommand` operation offered by the latter.

SmartLightingKit – part 2 (cont.)

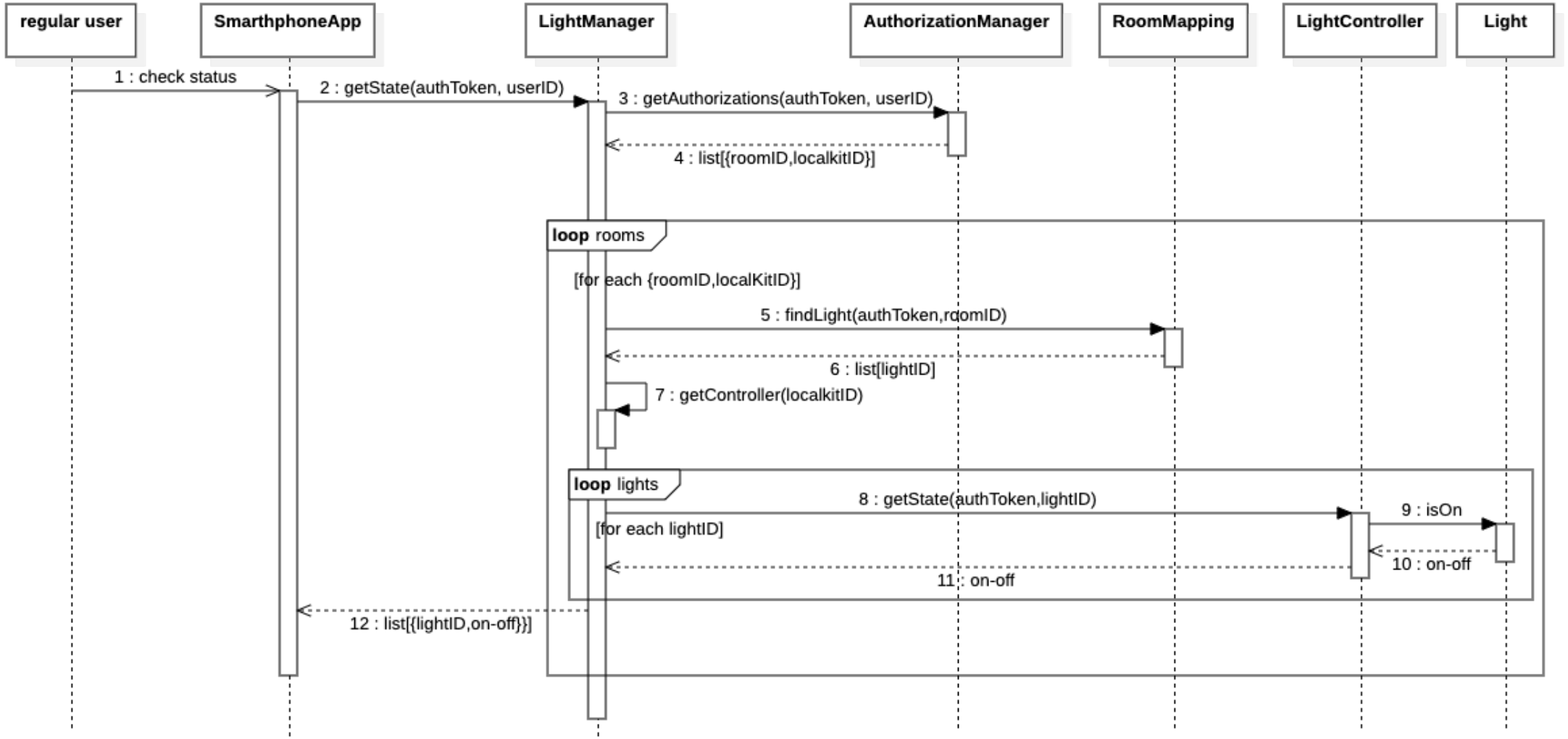
POLITECNICO MILANO 1863

- Q1:
  Analyze the operations offered by the components shown in the diagram and identify proper input and output parameters for each of them.
- Q2:
  Write a UML Sequence Diagram illustrating the interaction between the software components when a regular user wants to use the smartphone application to check whether he/she left some lights on (among the lights he/she can control).

Operations

- **APIGateway**
- *getLights*: input = userID; output = list[lightID]
- *getState*: input = userID; output = list[(lightID, state)]
- *setState*: input = userID, lightID, state; output = none
- **AuthI**
- *getAuthorizations*: input = userID; output = list[(roomID, localKitID)]
- **RoomI**
- *findLight*: input = roomID; output = list[lightID]
- **LocalControllerI**
- *switch*: input = lightID; output = none
- *getState*: input = lightID; output = state
- **LightI**
- *isOn*: input = none, output = True/False
- *switch*: input = none, output = none
- **SwitchI**
- *switchCommand*: input = none; output = none
- **Note**
- State = On/Off;
- All the operations of `APIGateway`, `AuthI`, `RoomI`, `LocalKitI` receive as input also the authentication token.

Sequence diagram

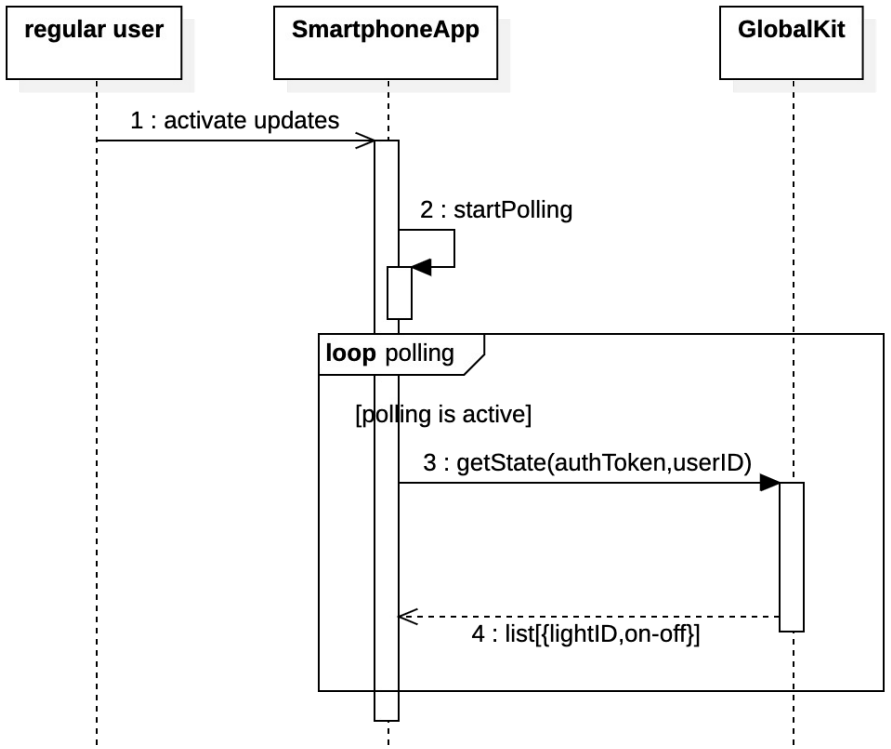


SmartLightingKit – part 3

POLITECNICO MILANO 1863

- Assume that, at a certain point, the following new requirement is defined:
  - **NewReq**: *"The smartphone application should allow users to activate/deactivate the receival of real-time updates about the state (on/off) of all the lights they can control."*
- Define a high-level UML Sequence Diagram to describe how the current architecture could accommodate this requirement.
- Highlight the main disadvantage emerging from the sequence diagram.

Realizing NewReq



- Problem: the current architecture does not support updates in push mode.
- The `SmartphoneApp` carries out a continuous polling process to retrieve the status of all the lights even in case it does not change.
- This propagates also internally to `GlobalKit` and the involved `LocalKits`, thus resulting in a potentially significant communication overhead.

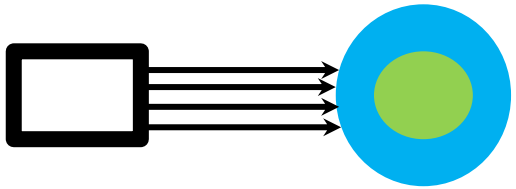
What did we learn from this exercise?

POLITECNICO MILANO 1863

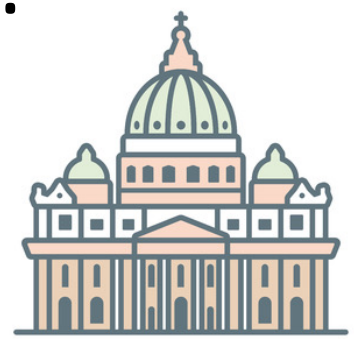
- From C&C structures we can:
  - Get an understanding of our system, infer which components are/can be replicated and how they could be allocated to execution environments
  - Identify and specify operations → essential to start development
  - Reason on the impact of changes
  - Identify new architectural options to address specific problems

What are the tools we have to develop and reason on software architectures?

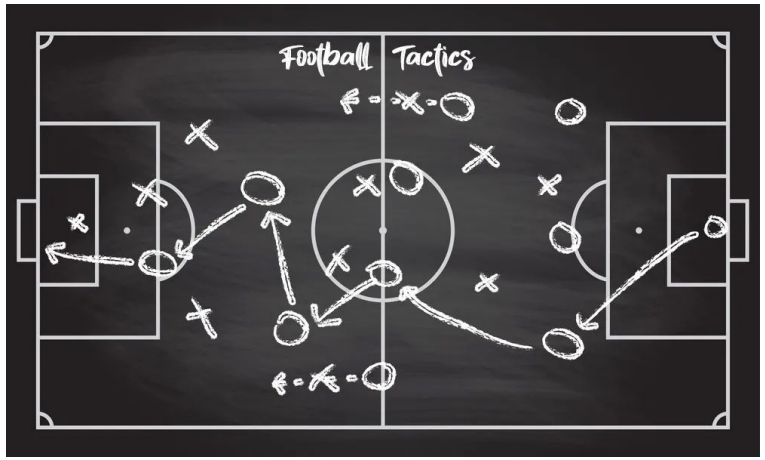
Standard guidelines

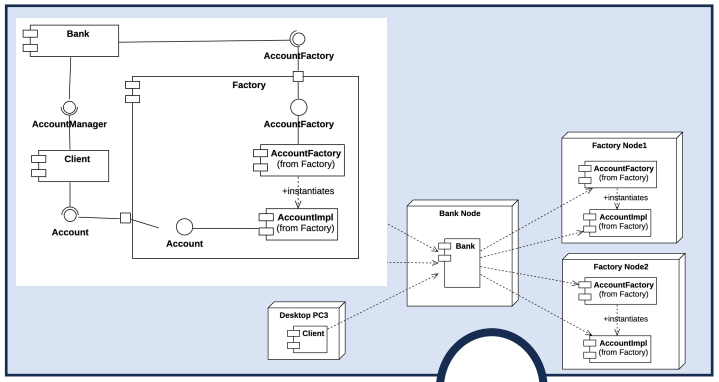


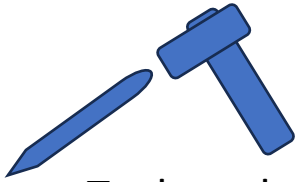
Design principles



Architectural styles



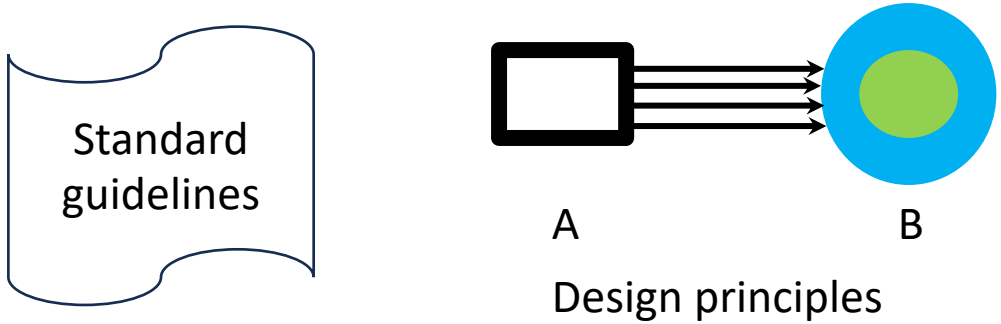




Tools and frameworks

Analysis approaches

What are the tools we have to develop and reason on software architectures?



- Watch the video available here https://polimi365-my.sharepoint.com/:v:/g/personal/10143828_polimi_it/EaE5ix1izNxGiLLtzBsr7GABI6M_BqvtBPeH4qtuVMOnhQ?e=hydFes
- Corresponding slides available here https://webeep.polimi.it/pluginfile.php/1084264/mod_folder/content/0/04b.video1_DesignDescriptionAndPrinciples.pdf?forcedownload=1
- Note: what we call “structures” in this lecture is called “views” in the video