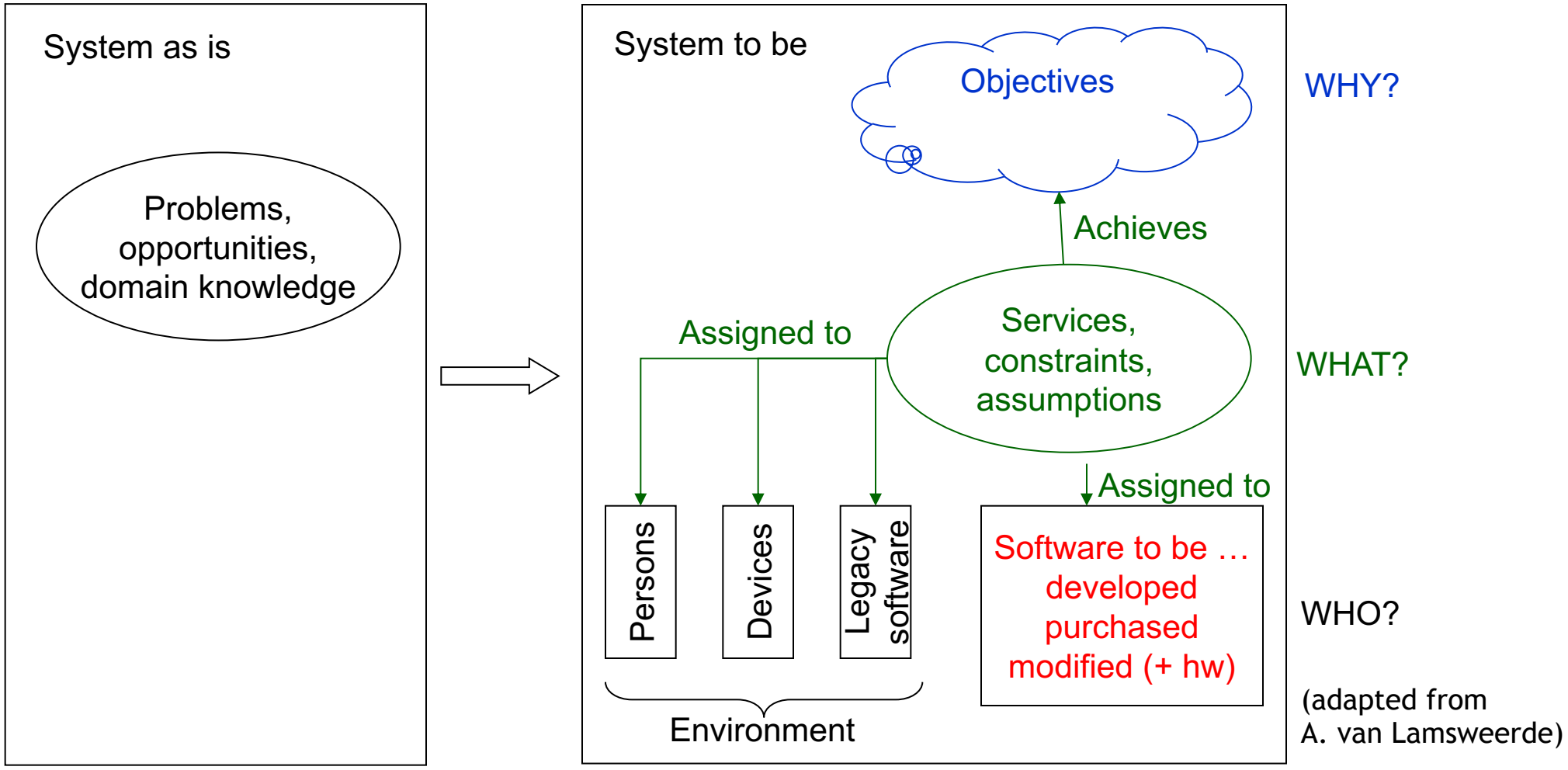
POLITECNICO
MILANO 1863

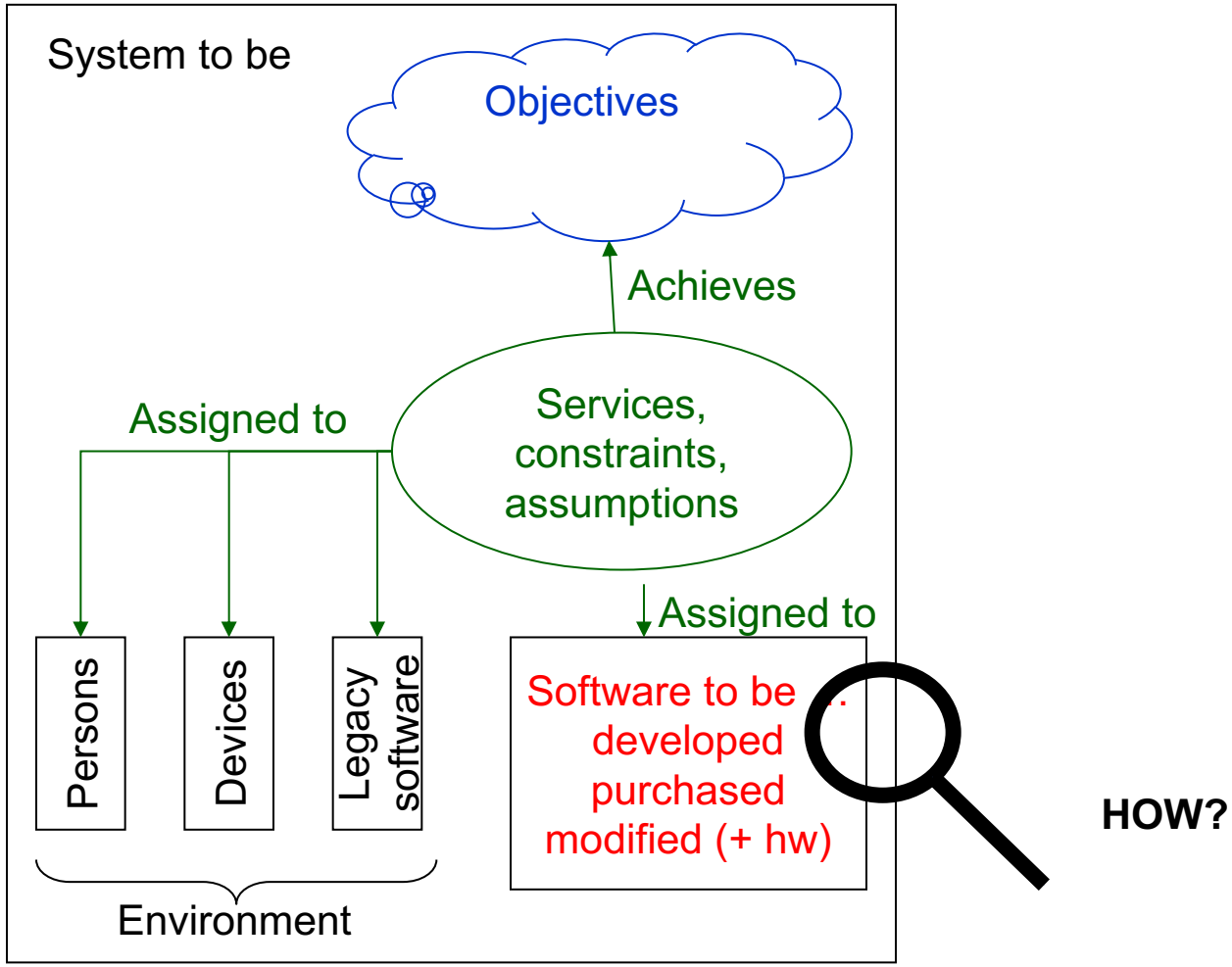
Software Design

Introduction

Focuses of requirement engineering



Focus of software design



HOW?

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.*

A simple example

https://www.codejava.net/java-se/networking/java-socket-server-examples-tcp-ip

```java
import java.io.*;
import java.net.*;
import java.util.Date;

/**
 * This program demonstrates a simple TCP/IP socket server.
 *
 * @author www.codejava.net
 */
public class TimeServer {

    public static void main(String[] args) {
        if (args.length < 1) return;

        int port = Integer.parseInt(args[0]);

        try (ServerSocket serverSocket = new ServerSocket(port)) {

            System.out.println("Server is listening on port " + port);

            while (true) {
                Socket socket = serverSocket.accept();

                System.out.println("New client connected");

                OutputStream output = socket.getOutputStream();
                PrintWriter writer = new PrintWriter(output, true);

                writer.println(new Date().toString());
            }

        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

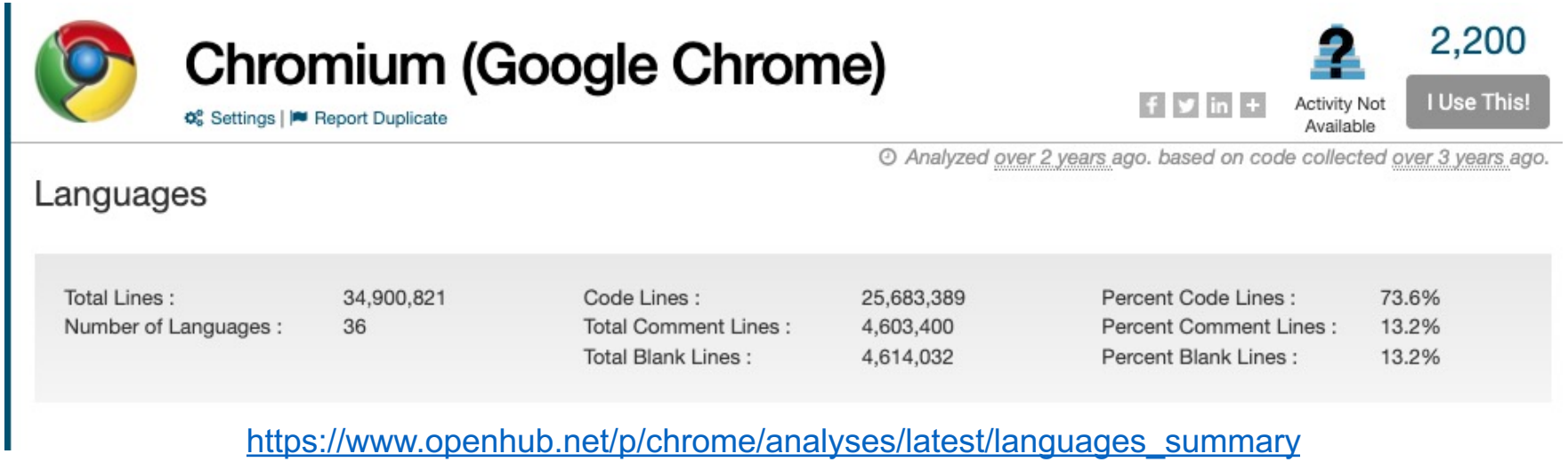
A simple example

https://www.codejava.net/java-se/networking/java-socket-server-examples-tcp-ip

```java
import java.net.*;
import java.io.*;

/**
 * This program demonstrates a simple TCP/IP socket client.
 *
 * @author www.codejava.net
 */
public class TimeClient {

    public static void main(String[] args) {
        if (args.length < 2) return;

        String hostname = args[0];
        int port = Integer.parseInt(args[1]);

        try (Socket socket = new Socket(hostname, port)) {

            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));

            String time = reader.readLine();

            System.out.println(time);


        } catch (UnknownHostException ex) {

            System.out.println("Server not found: " + ex.getMessage());

        } catch (IOException ex) {

            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Source code vs more abstract structures

- The source code is the ground truth!
- … but is it not always possible to look at it to understand how the system works?
- Try to understand the architecture of Chromium https://github.com/chromium/chromium ...



Chromium (Google Chrome)

Settings | Report Duplicate

Activity Not Available

2,200

I Use This!

Analyzed over 2 years ago. based on code collected over 3 years ago.

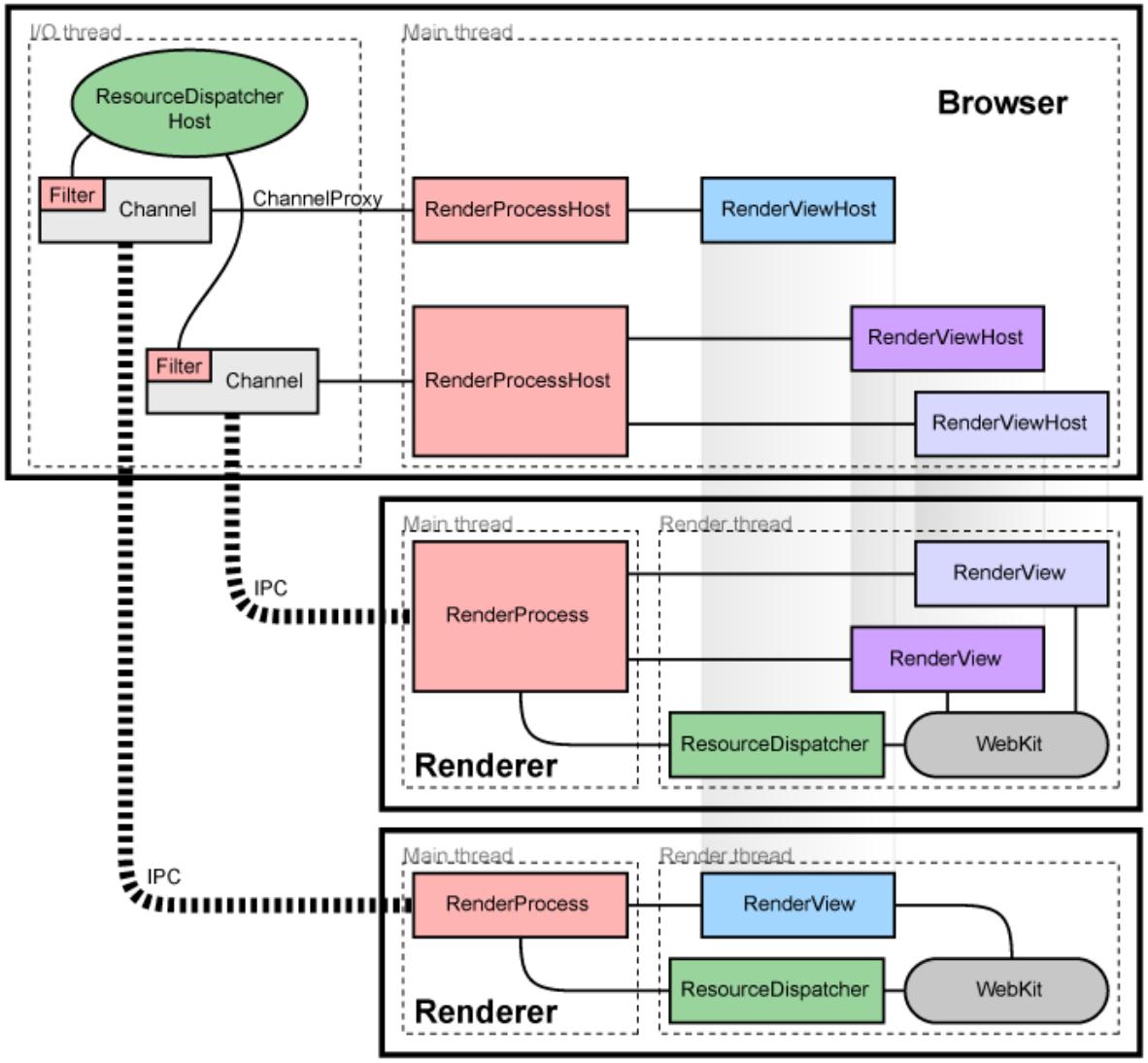
Languages

Total Lines :	34,900,821	Code Lines :	25,683,389	Percent Code Lines :	73.6%
Number of Languages :	36	Total Comment Lines :	4,603,400	Percent Comment Lines :	13.2%
		Total Blank Lines :	4,614,032	Percent Blank Lines :	13.2%

https://www.openhub.net/p/chrome/analyses/latest/languages_summary

Chromium Architectural overview

https://www.chromium.org/developers/design-documents/multi-process-architecture



Why Is Architecture Important?

POLITECNICO MILANO 1863

- Architecture is the vehicle for communication: internal (different teams), external (teams and stakeholders)
- Architecture manifests the earliest set of design decisions
  - Introduces constraints on implementation
  - Introduces organizational structure
  - Inhibits or enables quality attributes
- Architecture is a transferable abstraction of a system

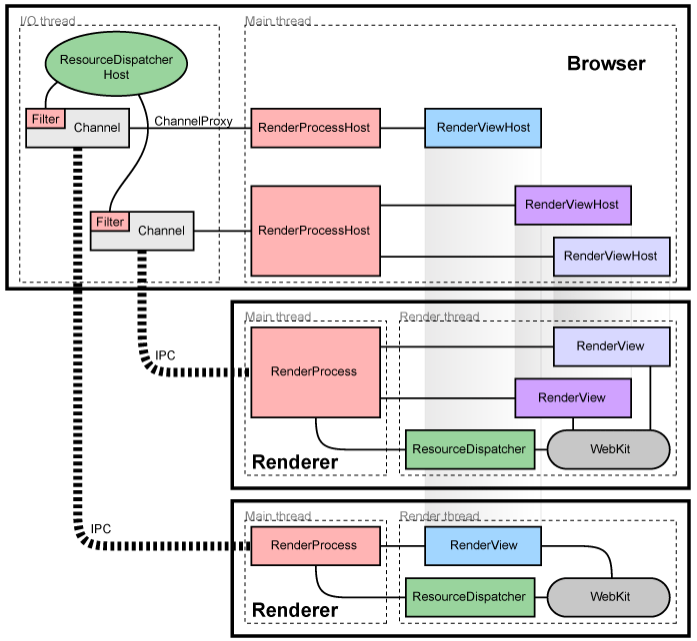
SE, Software Architecture, Hans van Vliet, ©2008

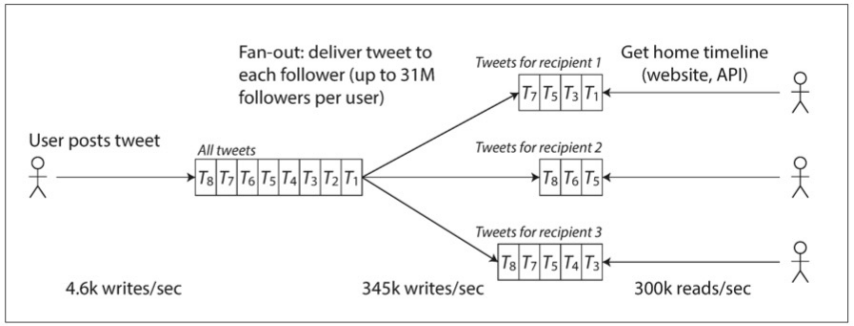
POLITECNICO
MILANO 1863

Architecture and multiple structures

Multiple views

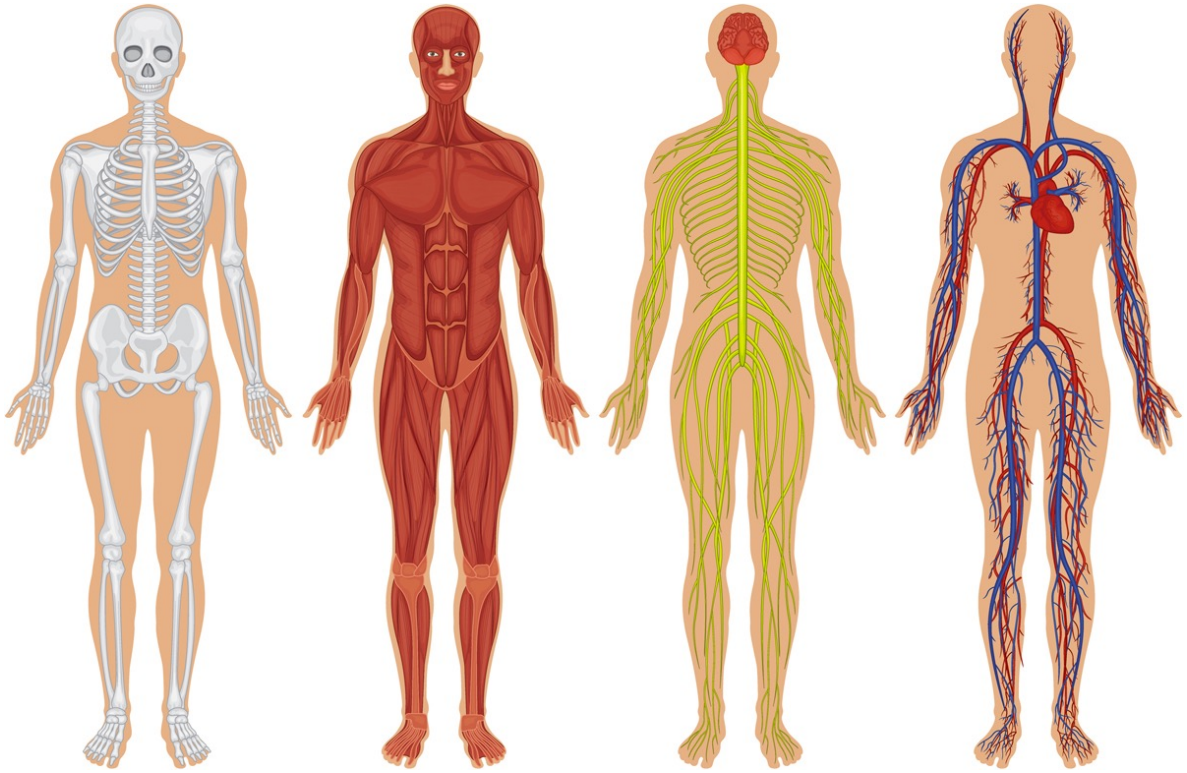
- Different concerns require focusing on different architectural structures





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



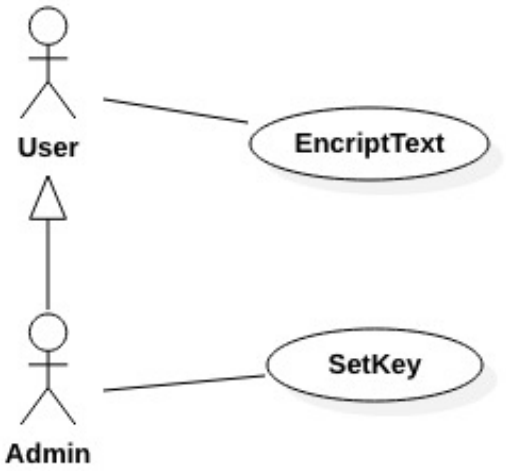
Example – Vigenère cipher

- Invented in 1553 by Giovan Battista Bellaso but misattributed to Blaise Vigenère
- A simple way to encrypt pieces of text
- Example
  - Original message: the quick brown fox jumps over 13 lazy dogs.
  - Encryption key: architecture
  - Encrypted message: tyg xcbgm ulfan wqe rnqrl imir 13 ccgg wsil.

a	r	c		h	i	t	e	c		t	u	r	e	a		r	c	h	
t	h	e		q	u	i	c	k		b	r	o	w	n		f	o	x	
t	y	g		x	c	b	g	m		u	l	f	a	n		w	q	e	

Example – Vigenère cipher

- Our goal: We want to offer an encryption service based on the Vigenère cipher such that users can enter a key and then have a set of strings encrypted
- R1: the system shall allow an admin user to input a key of minimum 6 characters and maximum 20 characters
- R2: the system shall allow all users to input a string to be encrypted. Upon insertion, if a key has been set, then the system shall return the corresponding encrypted string, otherwise, it shall return an error

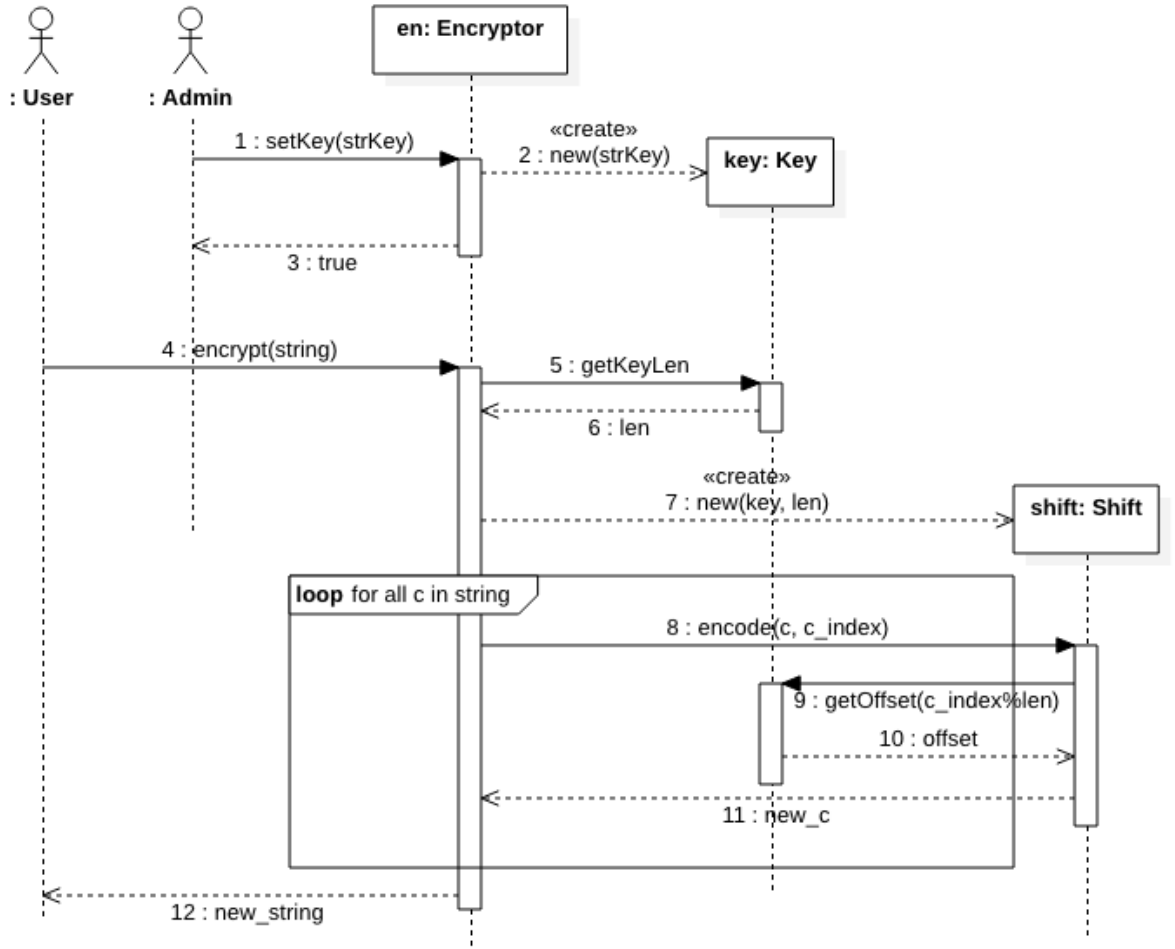


What are the components of our solution?

POLITECNICO MILANO 1863

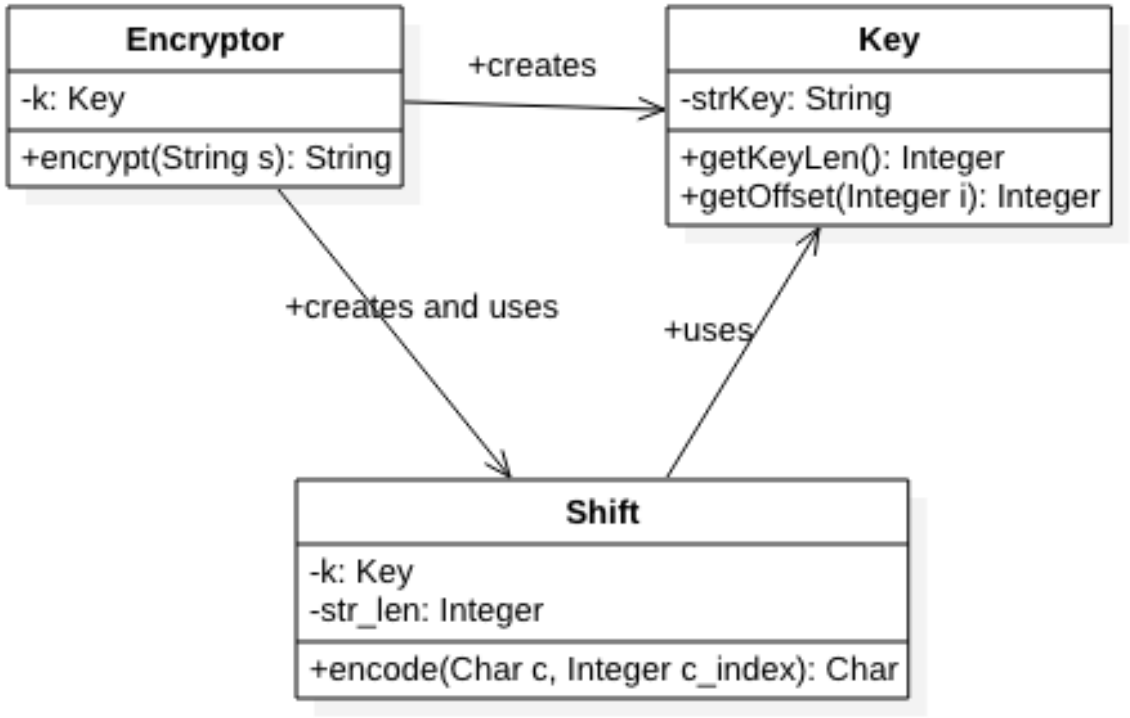
- Encryptor: this is the component that receives the requests from the users and coordinates the work of the other components
- Key: this could store the key defined by the admin and offer an operation that returns the number corresponding to the character in the specified position
- Shift: this could execute the transformation of an individual character based on the key

How do the components interact to offer the service?



- Questions that come to your mind?

What are the dependencies between modules implementing components



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, …

Component-and-connector structures: purpose

- 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

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

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

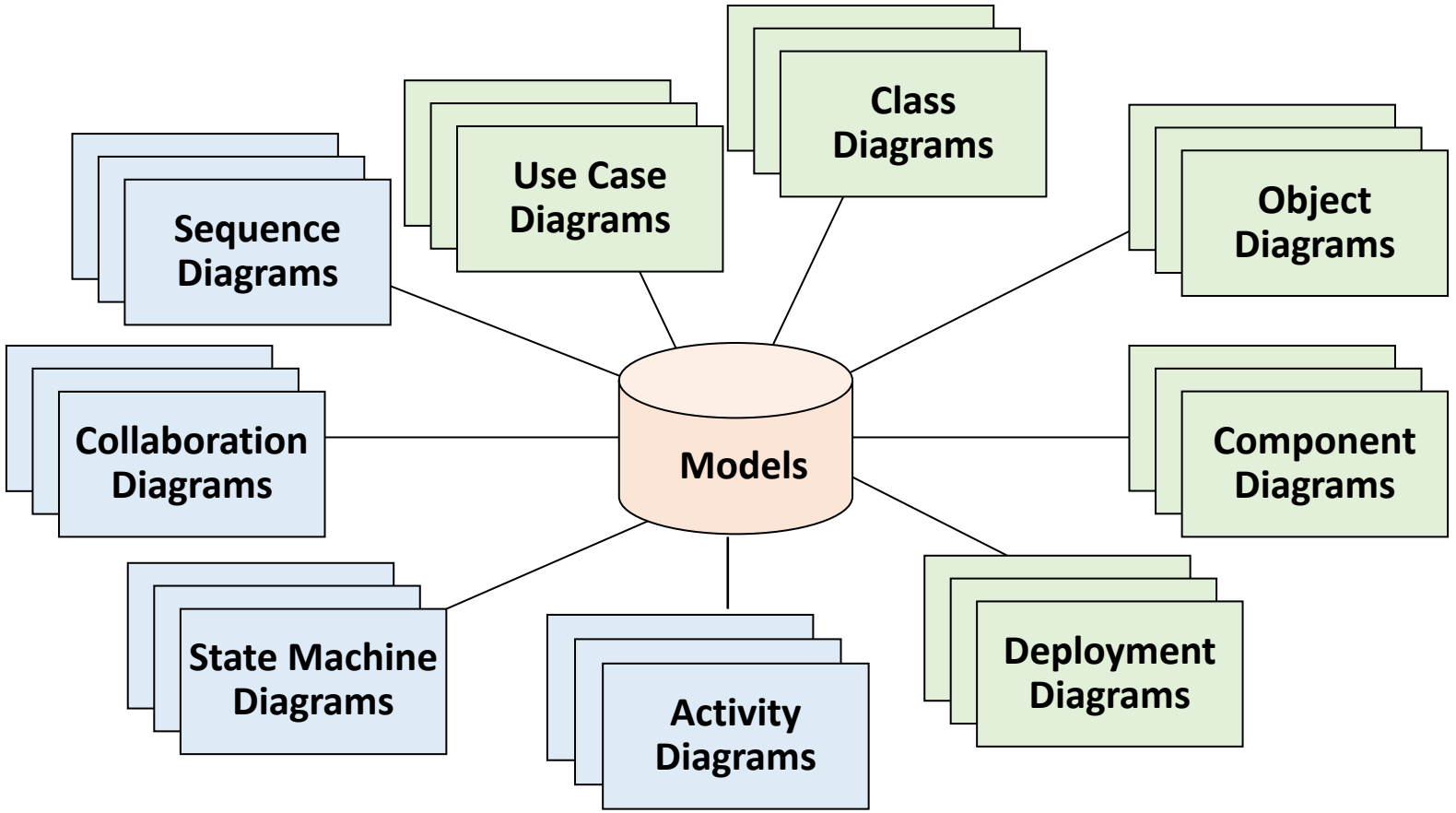
POLITECNICO MILANO 1863

- 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

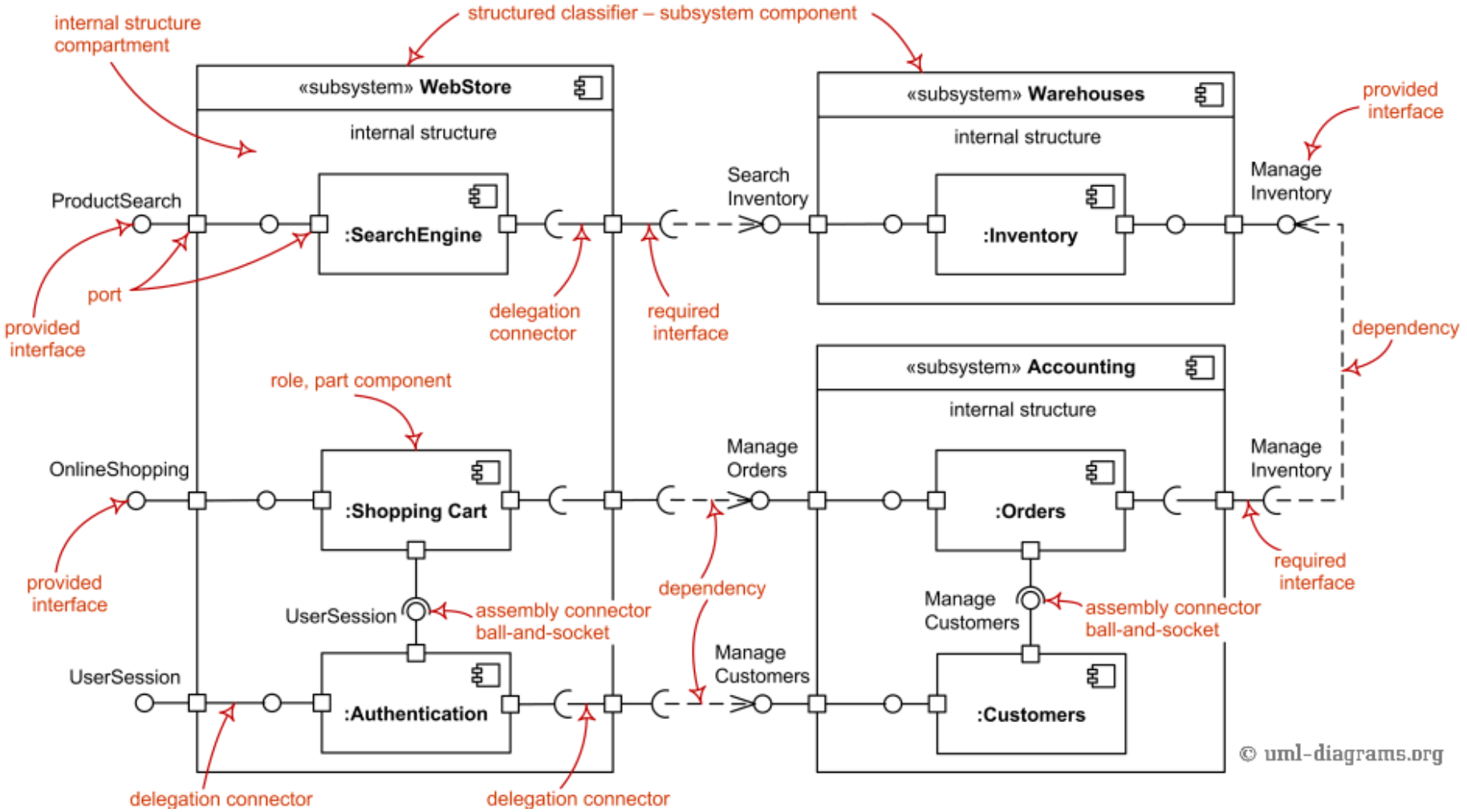
POLITECNICO
MILANO 1863

Software design descriptions and UML

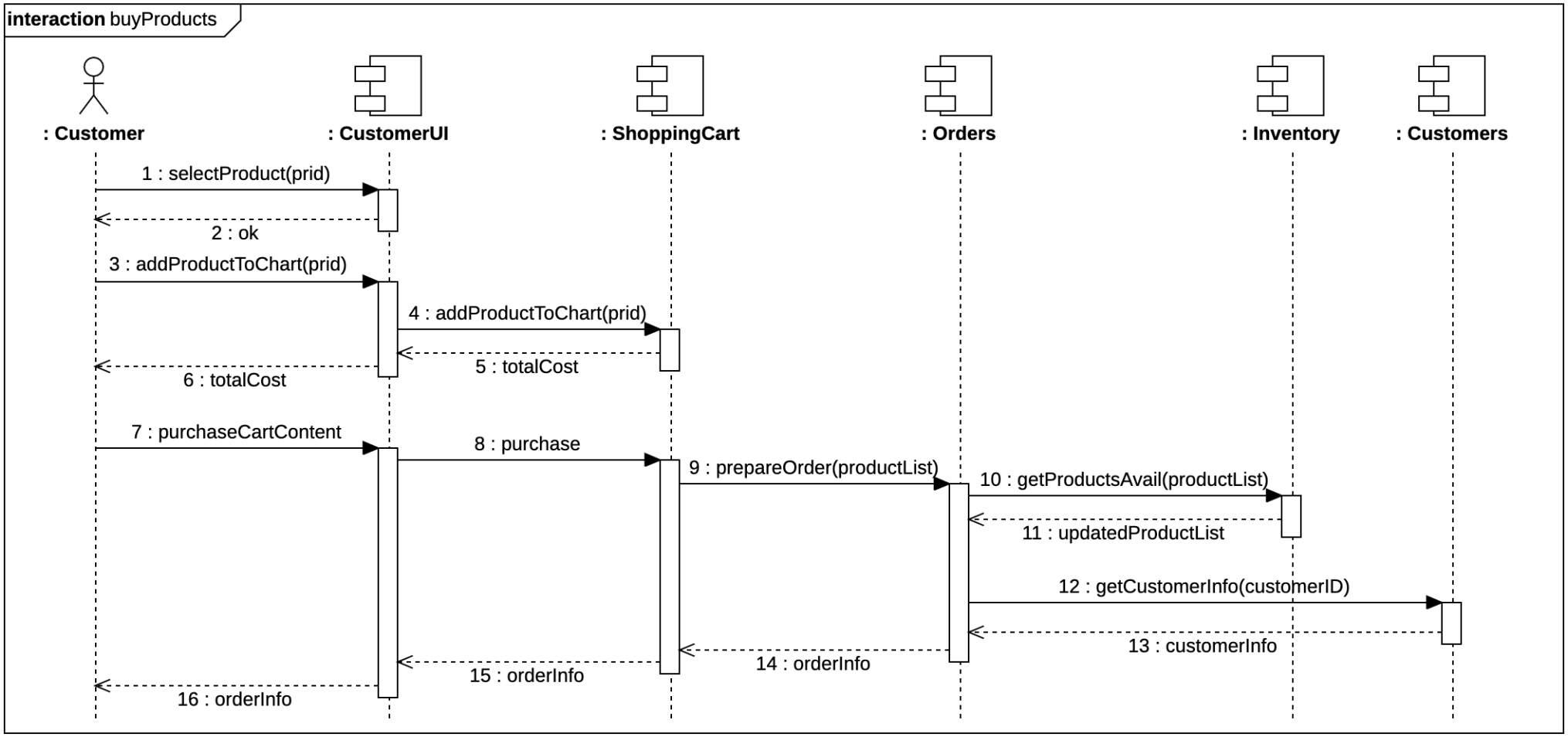
UML Models, Views, and Diagrams



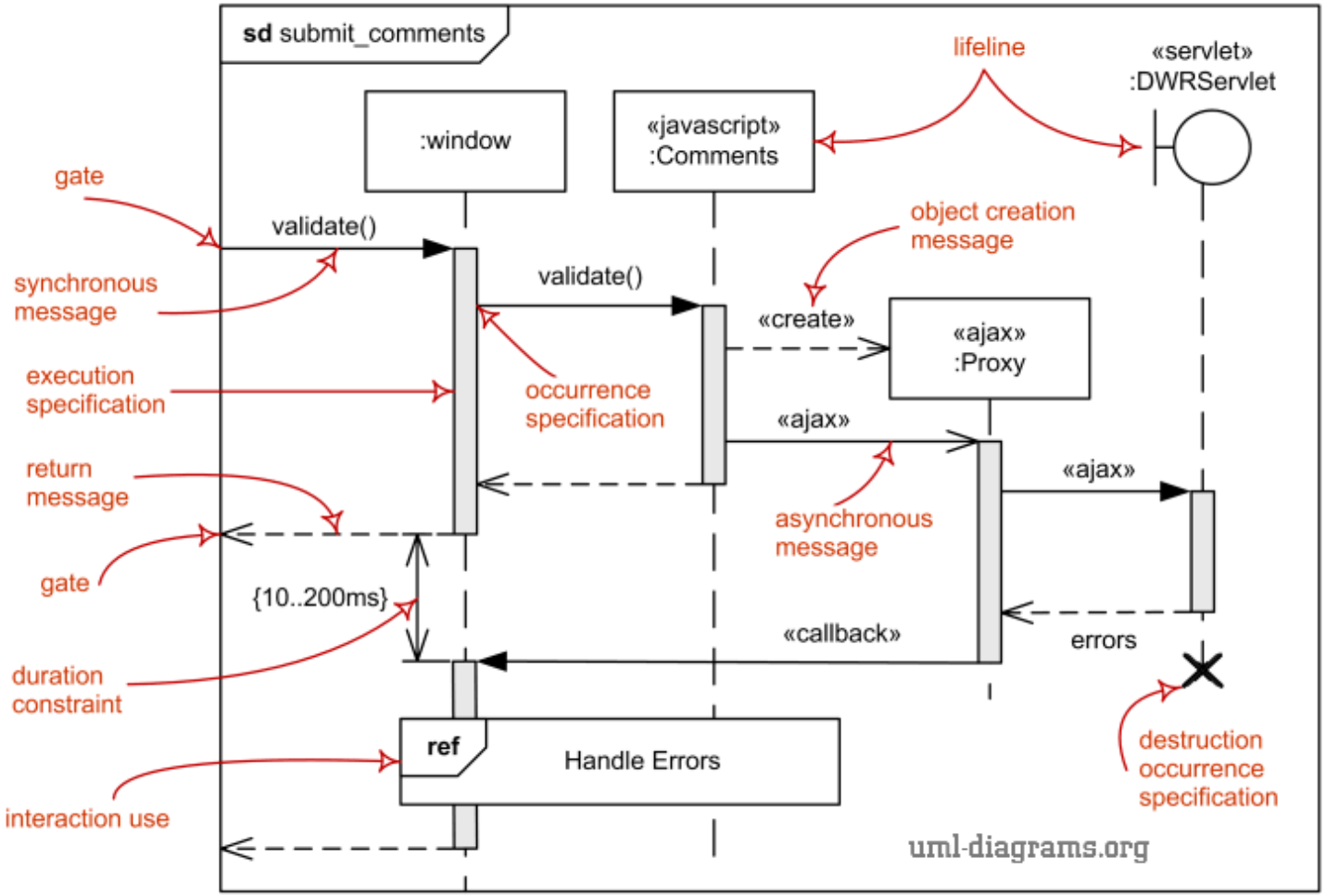
Component Diagram (C&C structure)



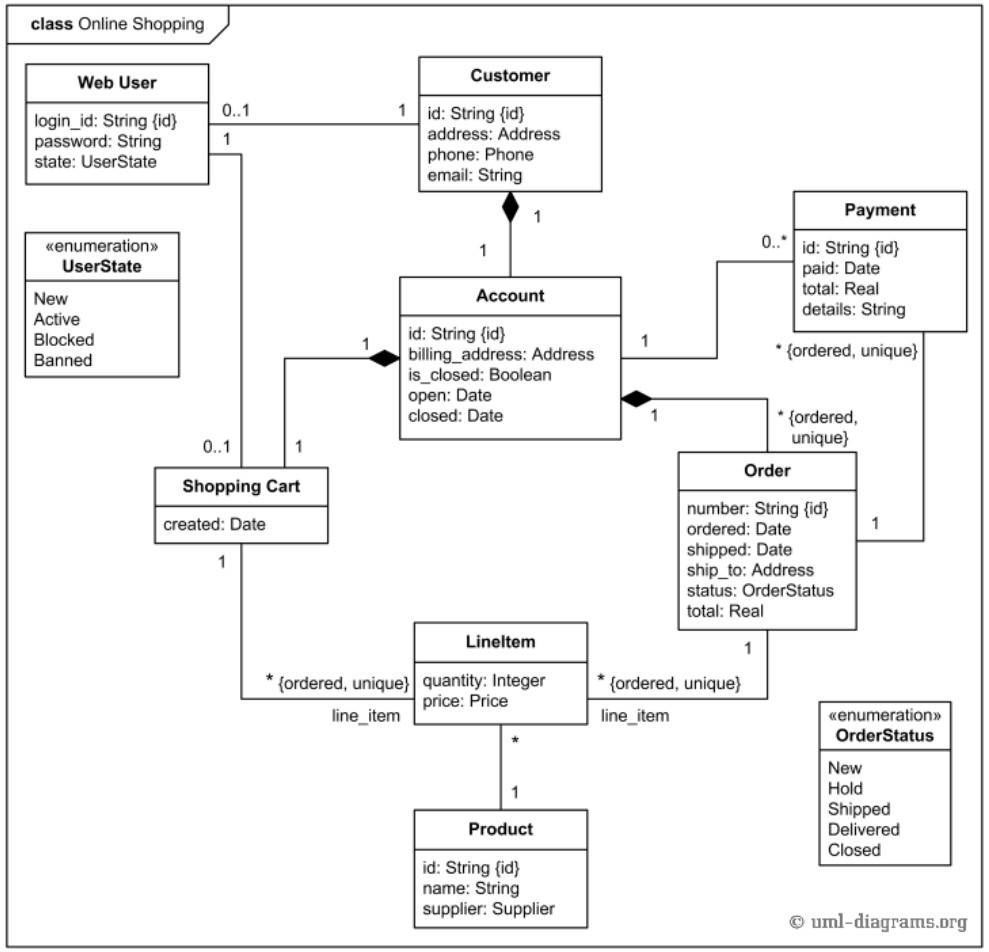
Sequence diagrams (C&C structure)



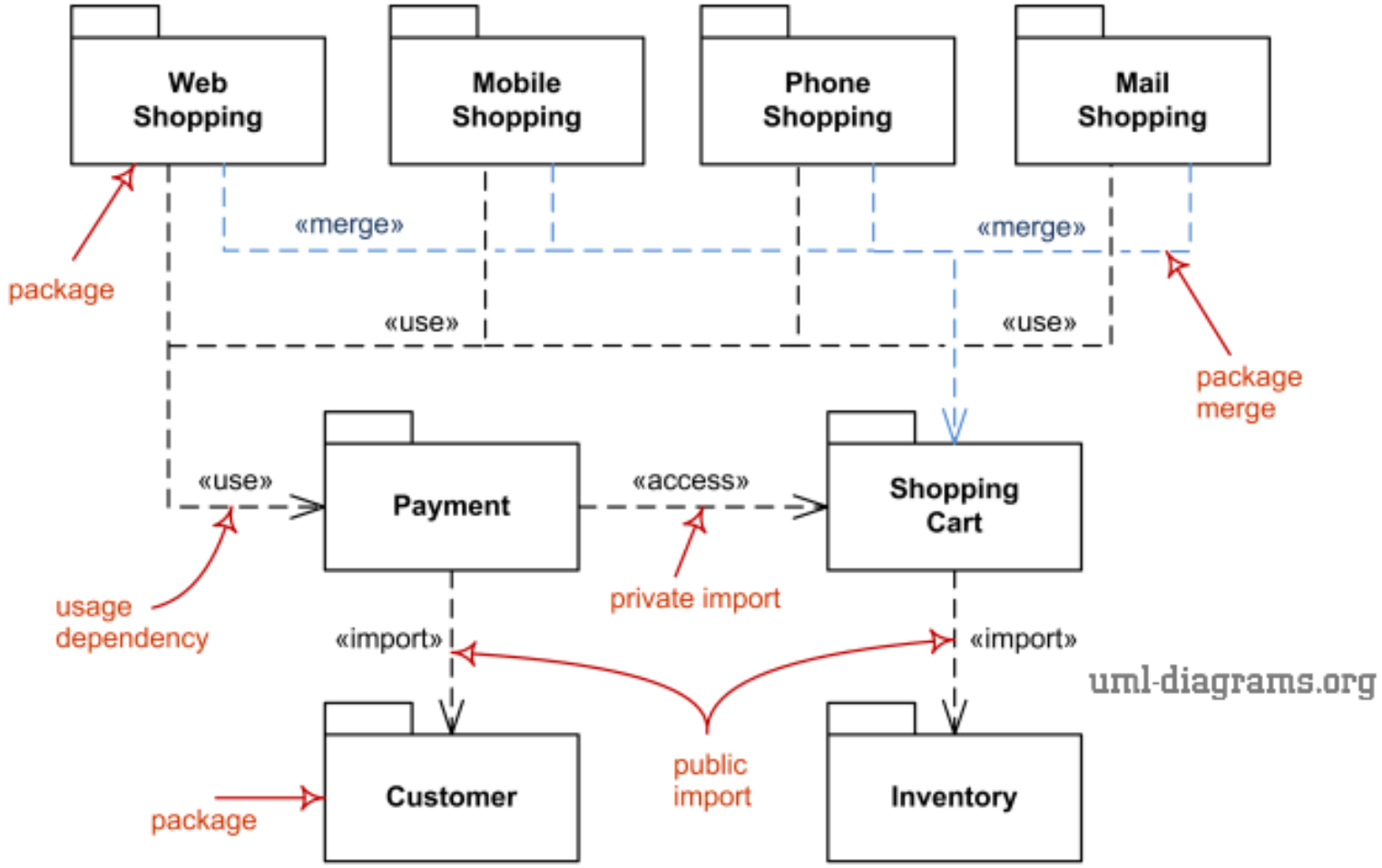
More on “Architectural” Sequence Diagram



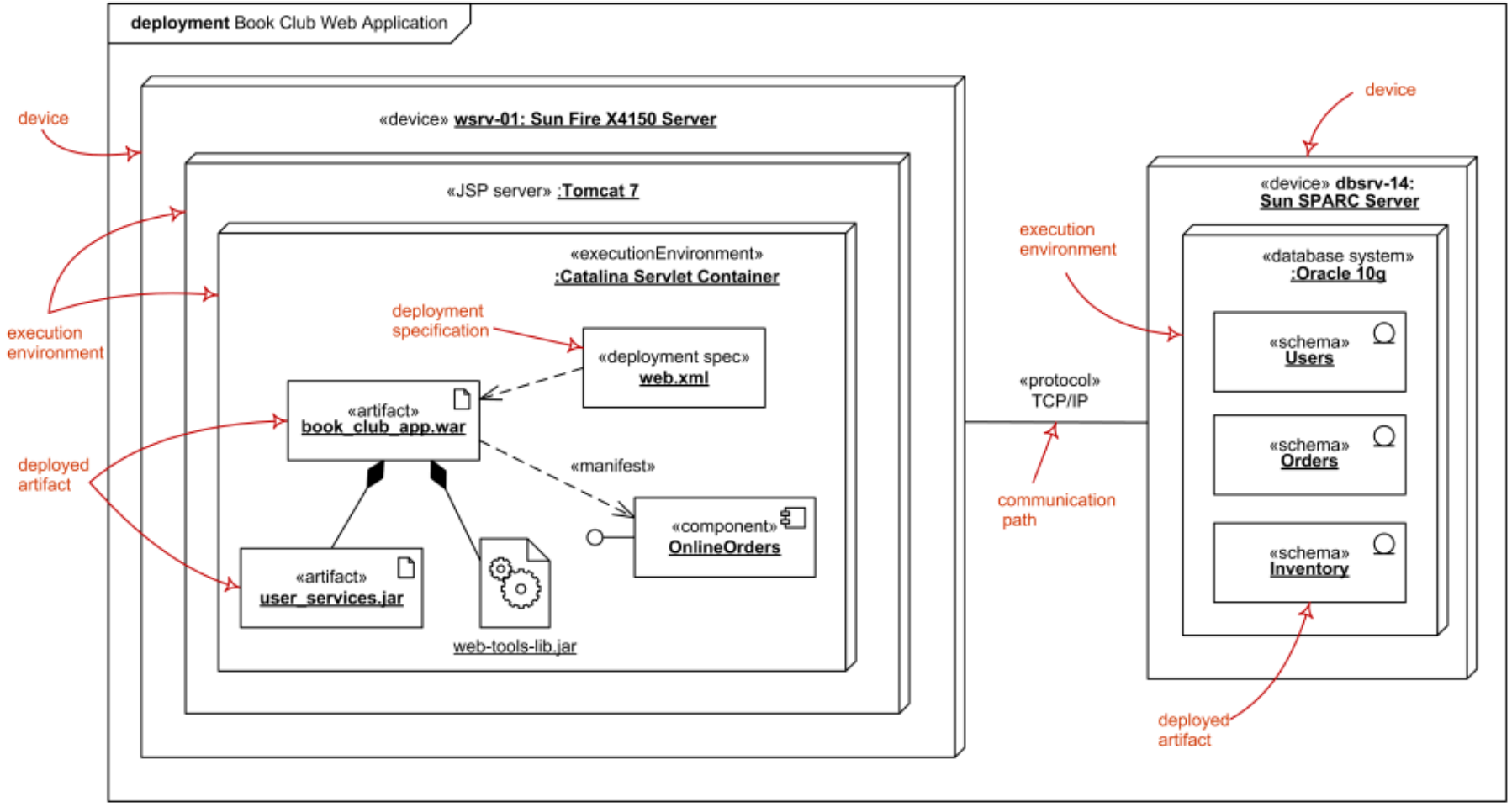
Class diagram (module structure)



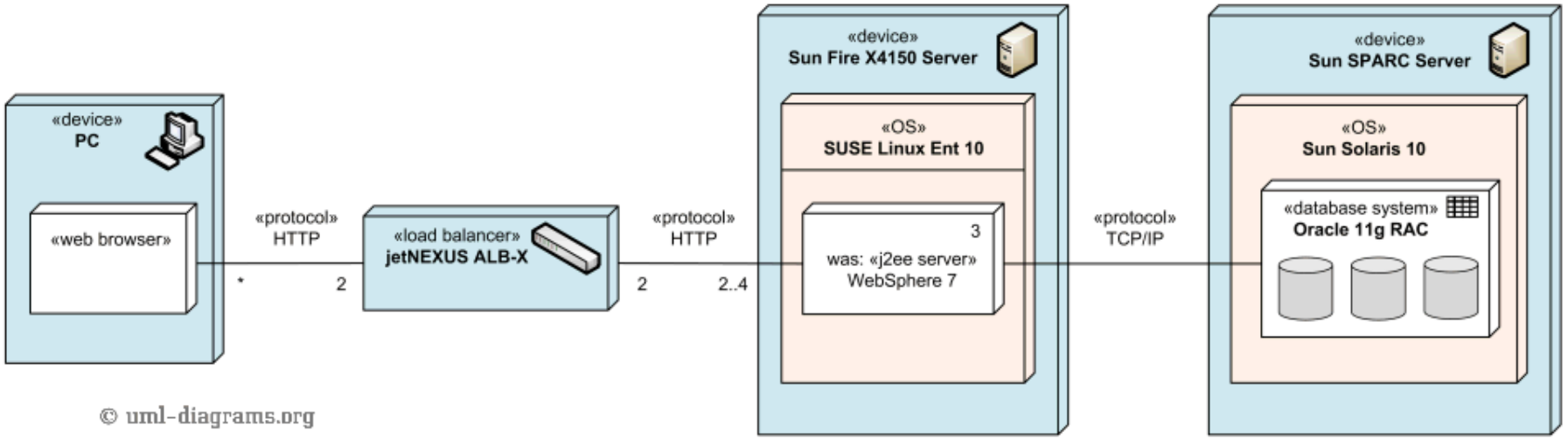
Package diagram (module structure)



Deployment diagram (allocation structure)



Deployment diagram (allocation structure)



Deployment Diagram

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

Exercise: Reasoning on architectural documentation

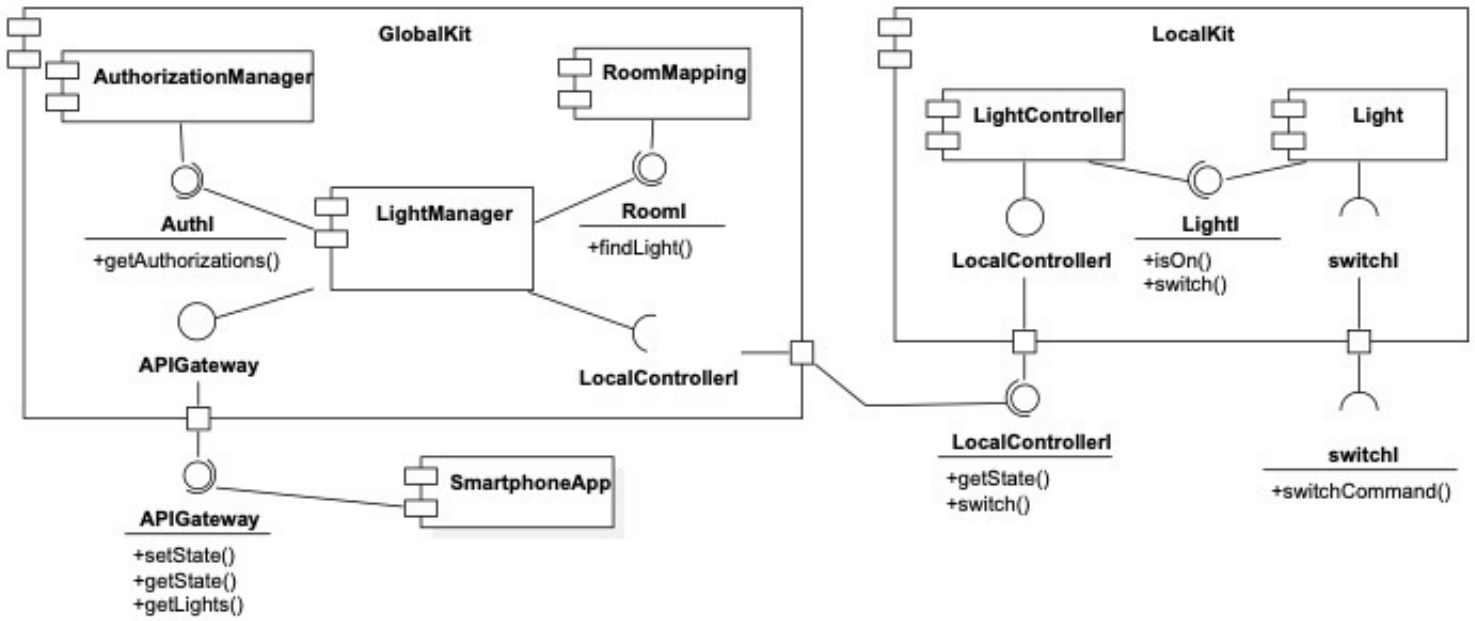
SmartLightingKit

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

- 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

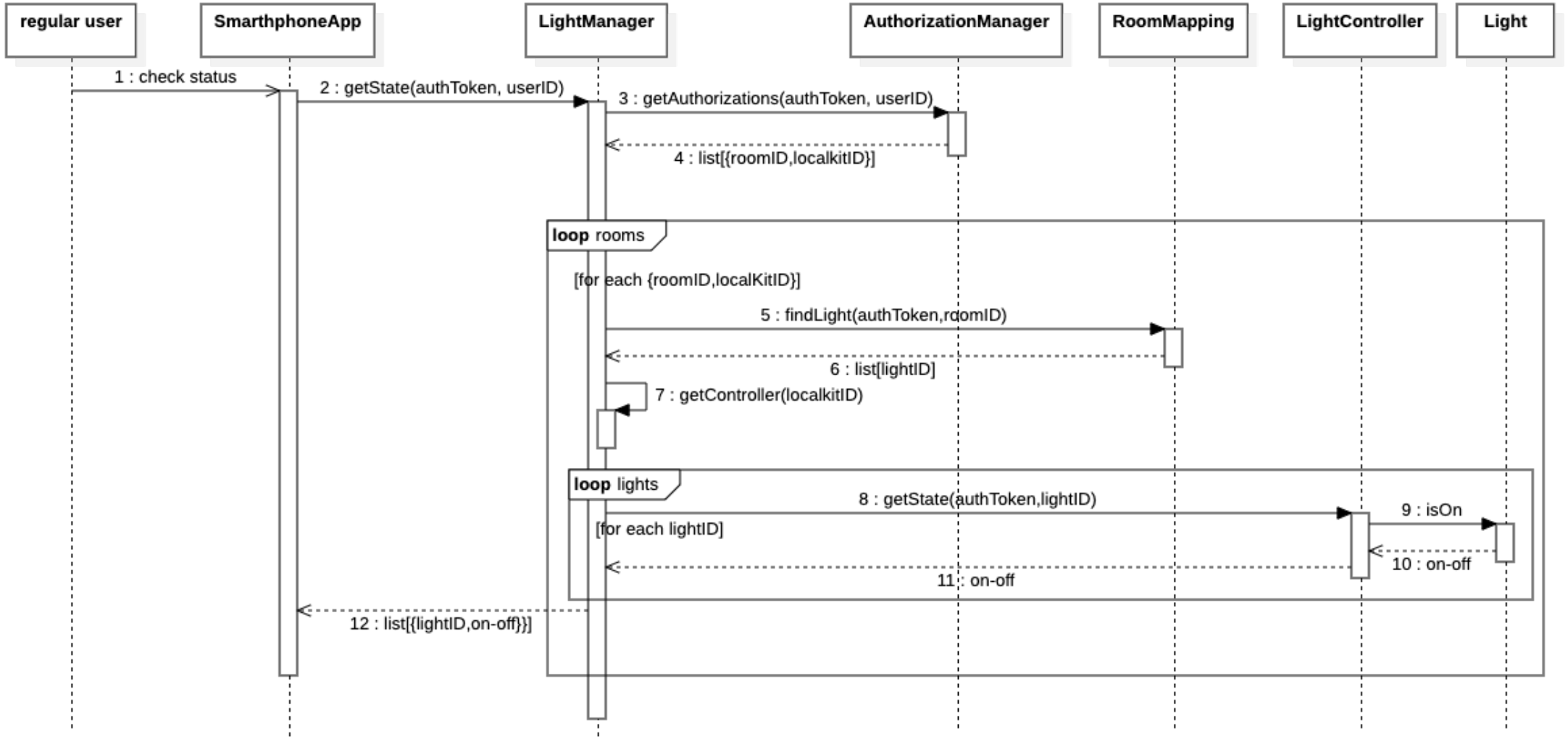
- Q1:
  Analyze the operations offered by the components shown in the diagram and identify proper input and output parameters for each of them.
- Q2:
  Write a UML Sequence Diagram illustrating the interaction between the software components when a regular user wants to use the smartphone application 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

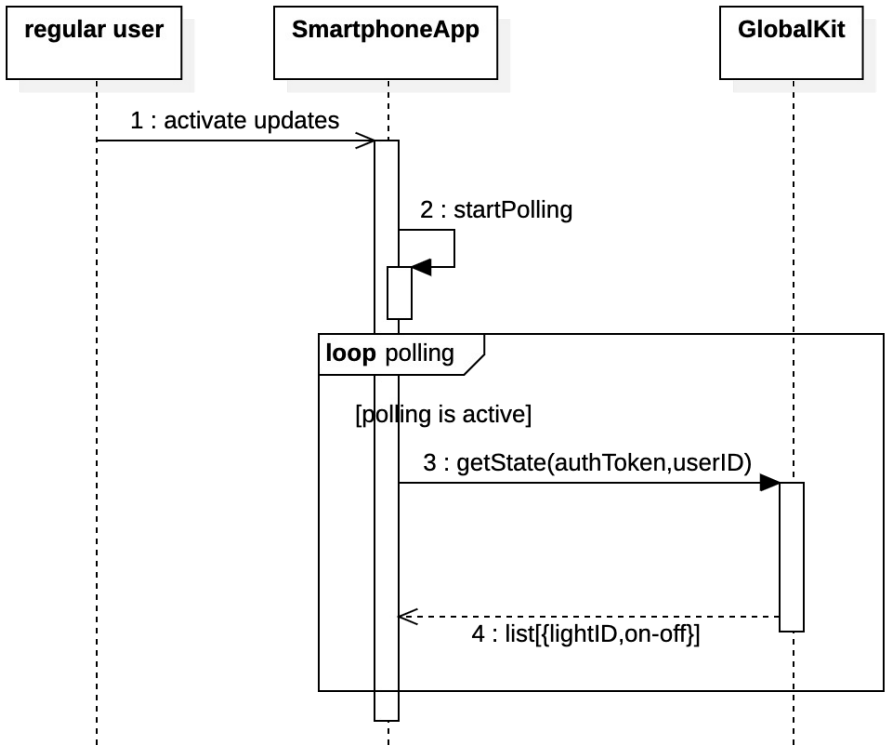


SmartLightingKit – part 3

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