Composite Architecture, PDOM

Lecture 10
4 June 2026

Slides created by
Prof Amir Tomer
tomera@cs.technion.ac.il

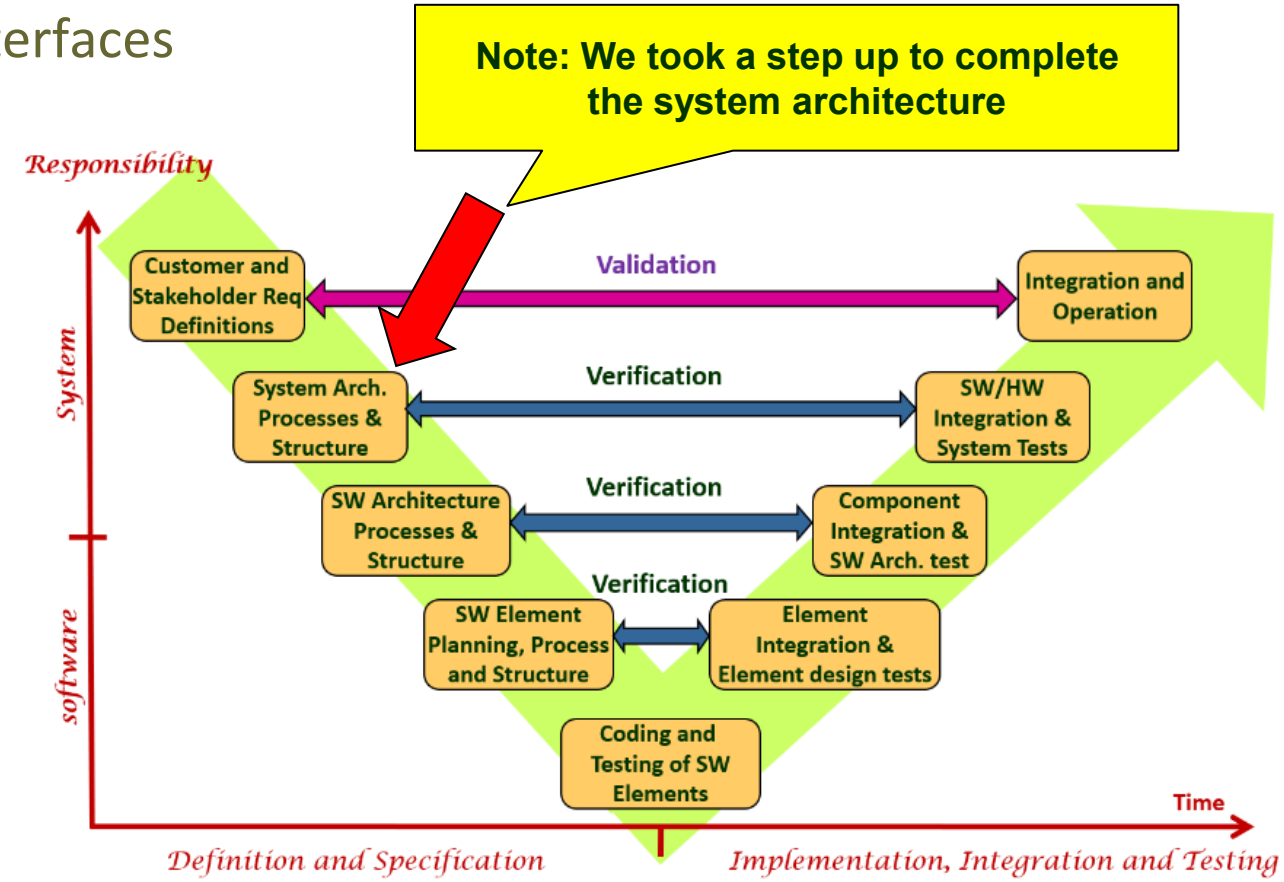
Picture Source: http://www.techsoft3d.com/developers/technical-documentation/siemens-parasolid/

Topics for Today

- Composite Architecture
- PDOM

Building Composite Architecture

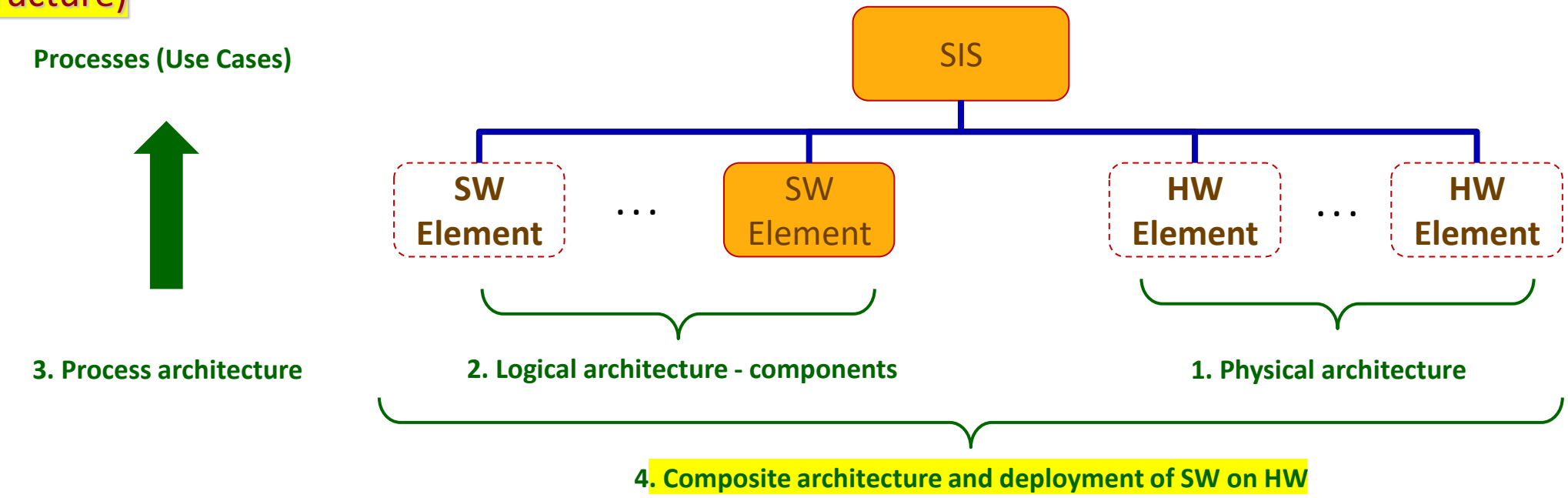
- Our goals:
  - Integrate the software and hardware architectures
  - Connect physical interfaces to logical interfaces
- Inputs:
  - Component Diagram
  - Deployment diagram
- Outputs:
  - Composite diagram
  - Build and deployment plans for software on the hardware



System Architecture: SIS architecture includes

Reminder

1. Physical architecture: Hardware components and physical connections – static model (structure)
2. Logical architecture: Software components and logical connections – static model (structure)
3. Process architecture: Implementation of processes via interaction between components - dynamic model (behavior)
4. Composite architecture: Implementation of logical connections via physical connections – static model (structure)

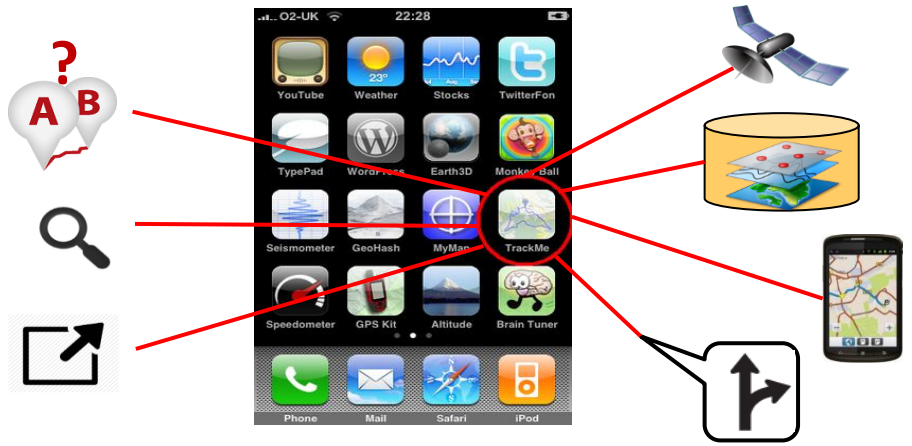


Functional interfaces via physical interfaces

Smartphone physical interfaces

Functional (logical) of navigation app

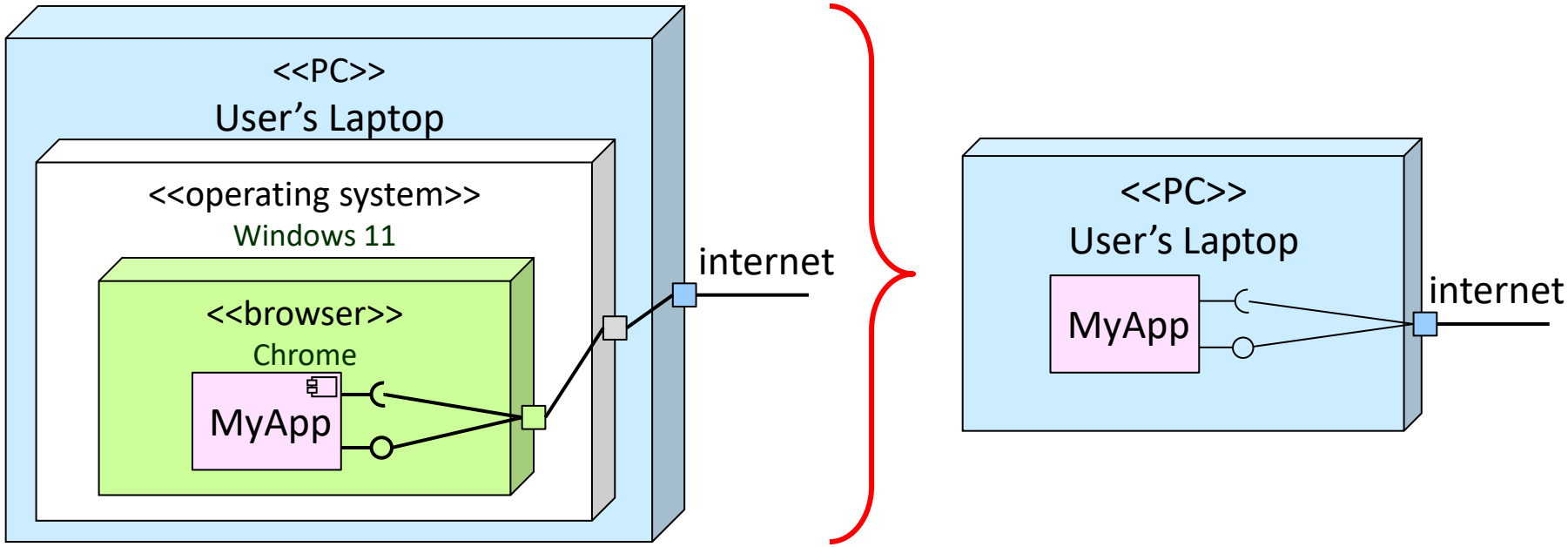
Provided *Interfaces*



Required Interfaces

Connection between functional and physical interfaces

- Sometimes one or more execution environment layers stand between functional and physical interfaces
  - Passage through each layer requires a specialized mechanism
  - The passage point is called a “port” and is shown with a small square
  - For simplicity, we might compress a few ports into a single general one



Delegation

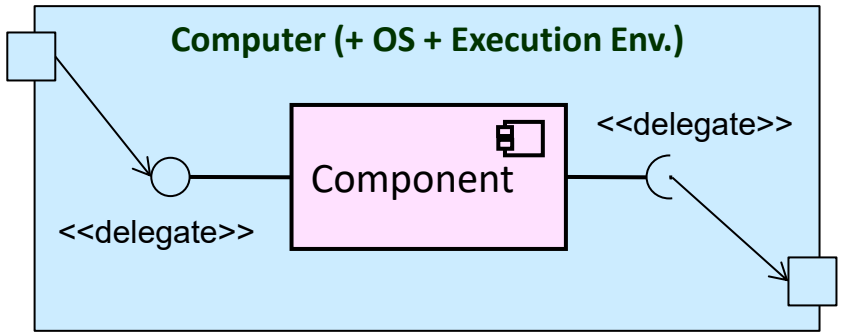
An internal interface can't directly contact the outside environment → it needs a **port** to represent it

Delegation from Provided interface

- The component provides the service to the external environment via the port
- Requests arrive at the port and are forwarded to the appropriate interface

Delegation from a Required interface

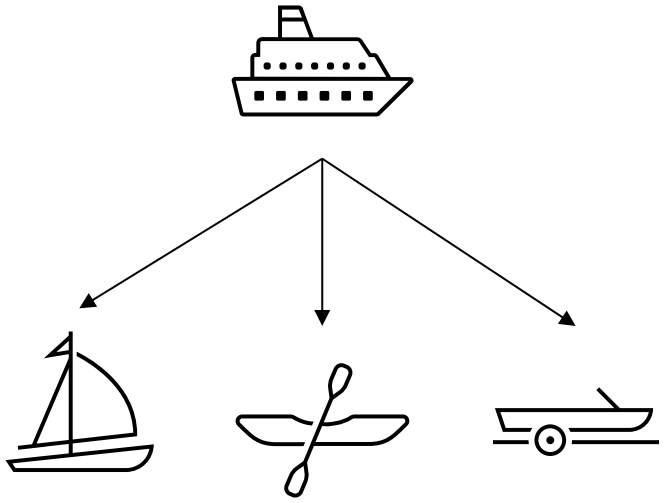
- The component receives the service from the external environment via the port
- Requests for the service leave via the port to the external environment



Difference between Logical and Physical architectures

Physical

- Physical architecture is built for a specific system
- Each configuration has its own physical architecture

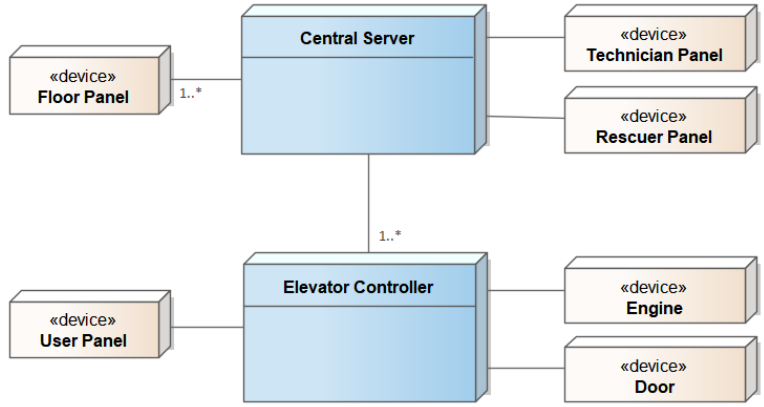


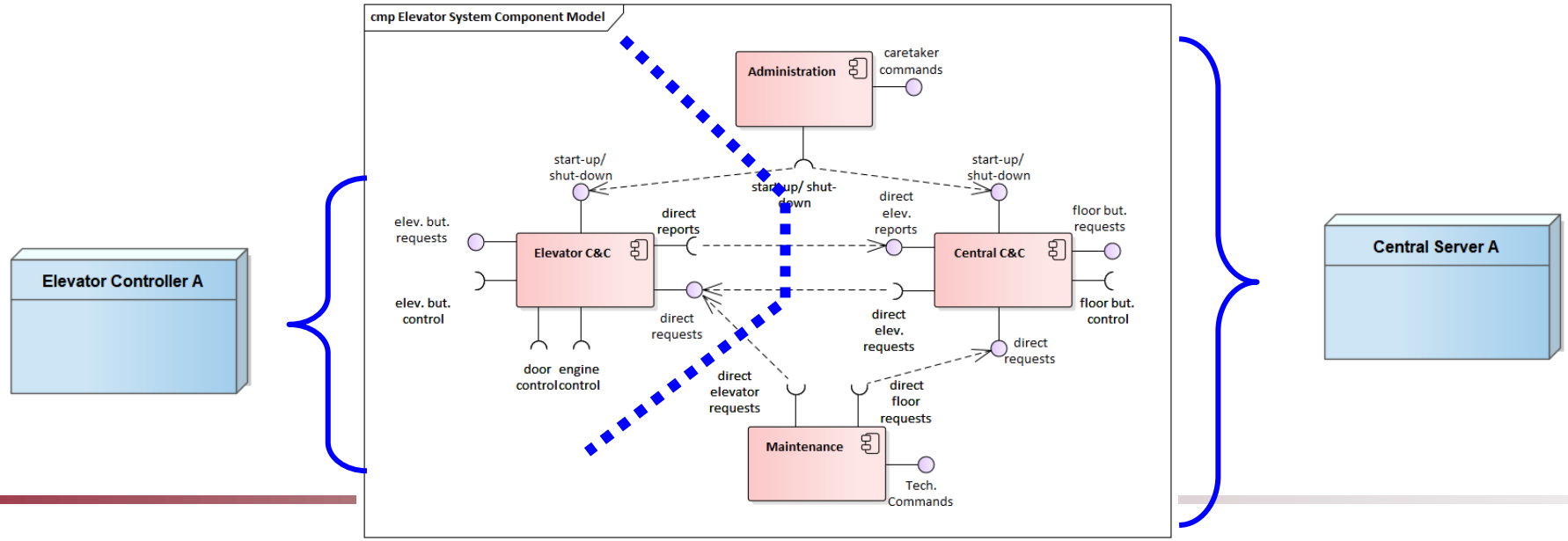
Logical

- Logical architecture is more generic
- Can use the same components for many systems
  - Many “builds”
- Connections between interfaces are implemented based on its physical location
  - Direct connection: components in same software (e.g. included, compiled)
  - Dependency: different software, same computer (e.g. DLL)
  - Via port: different computers, different software (e.g. client-server)

Building composite architecture: Step 1

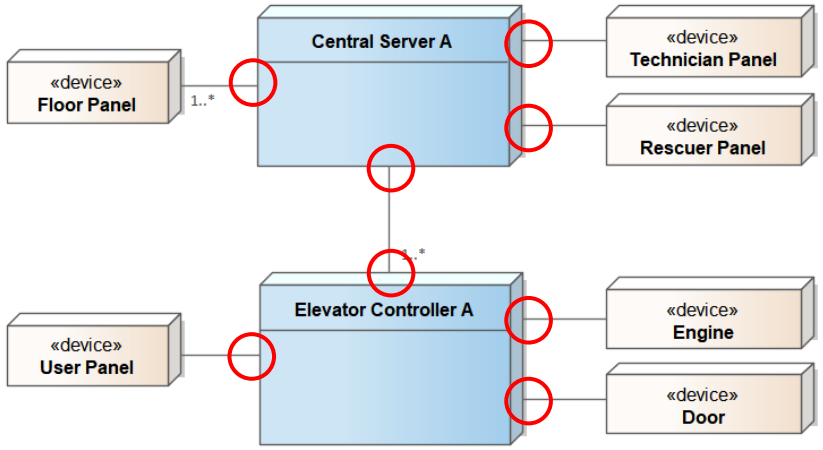
- Choose a physical architecture and decide how to break the components between the computers
- Elevator physical architecture A

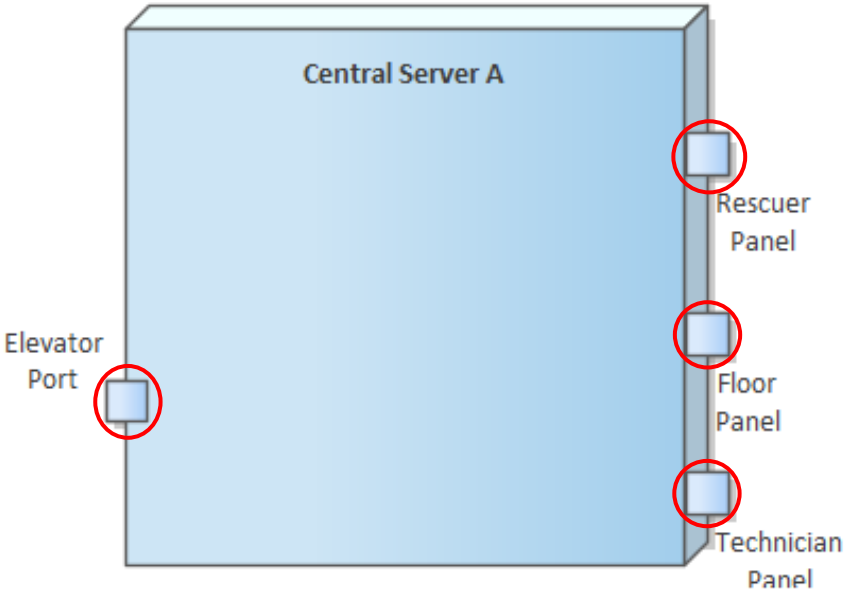


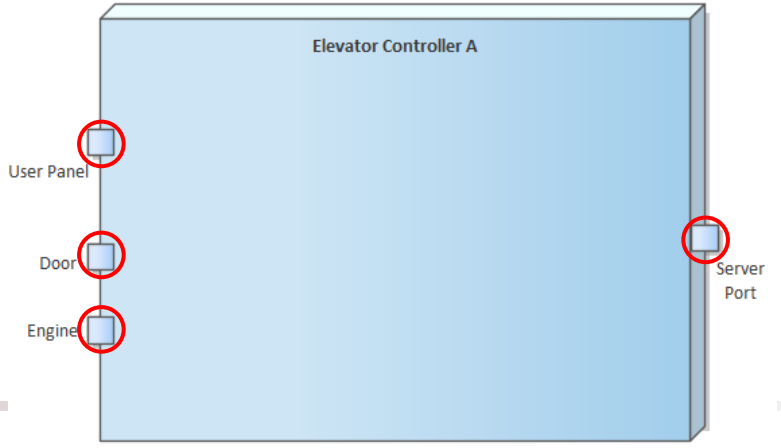


Building composite architecture: Step 2

- Create the composite diagram (can also do one diagram per computer)
  - Put the computers from the architecture in
  - For each physical connection, add a port

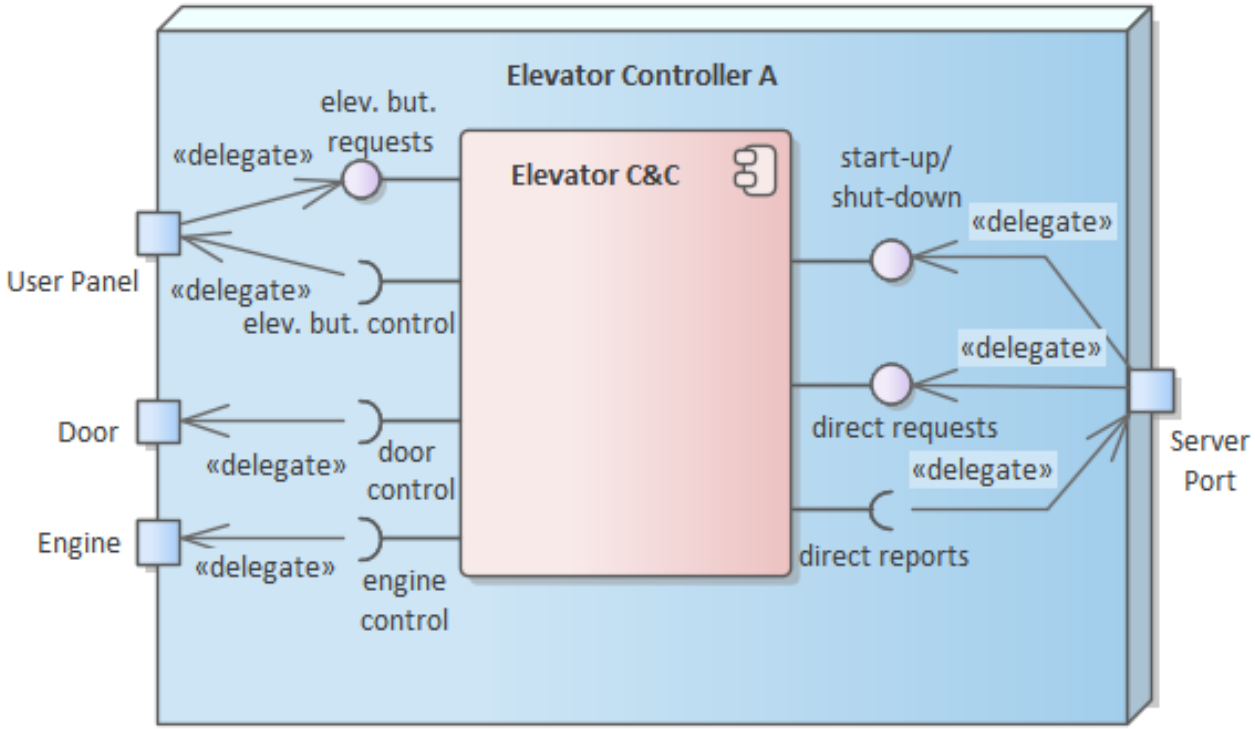




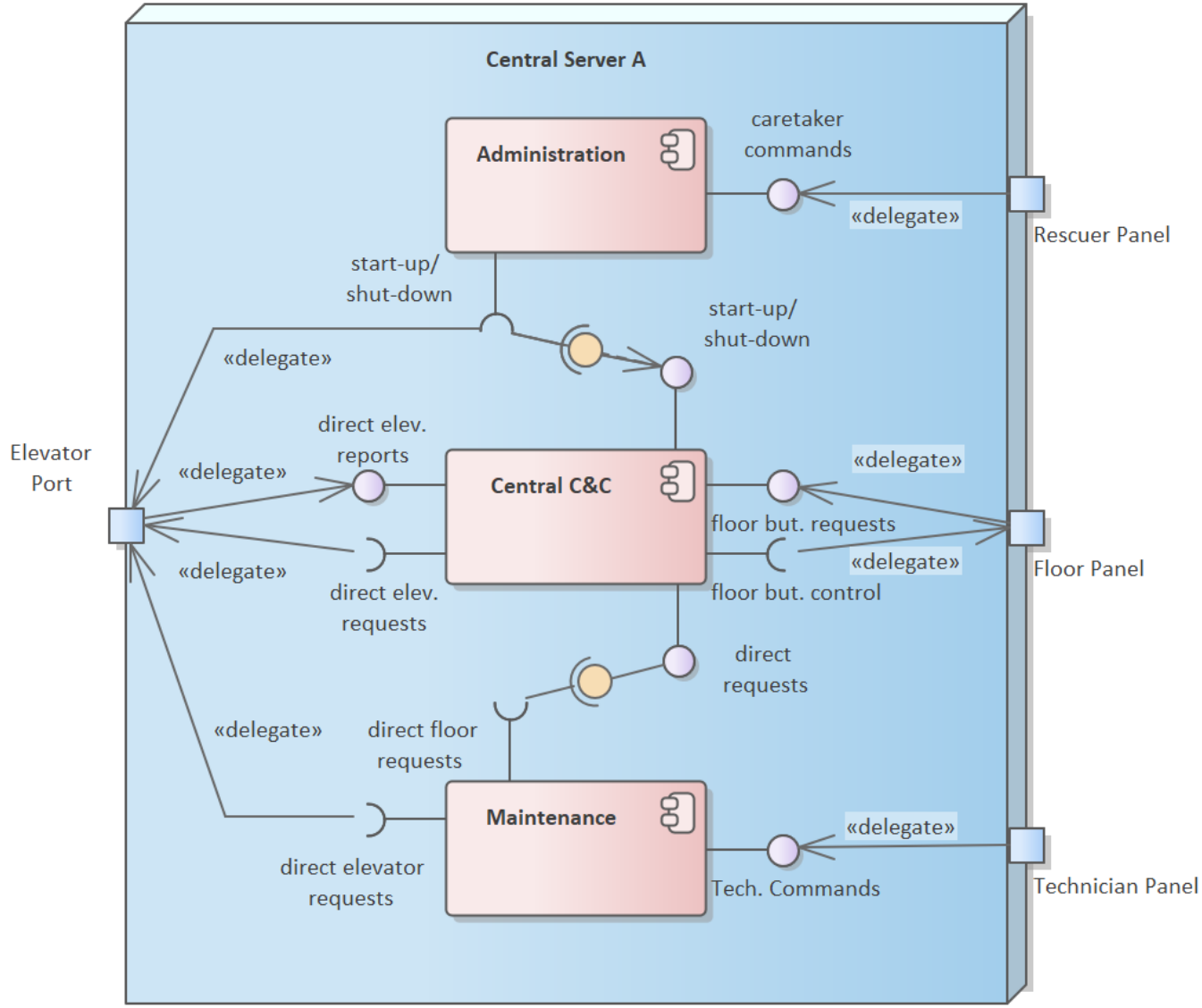


Building composite architecture: Step 3.1

- Place the functional components in the physical computers
- Connect the free interfaces to ports via <<delegate>> connections
- Connect internal dependencies with assembly connector



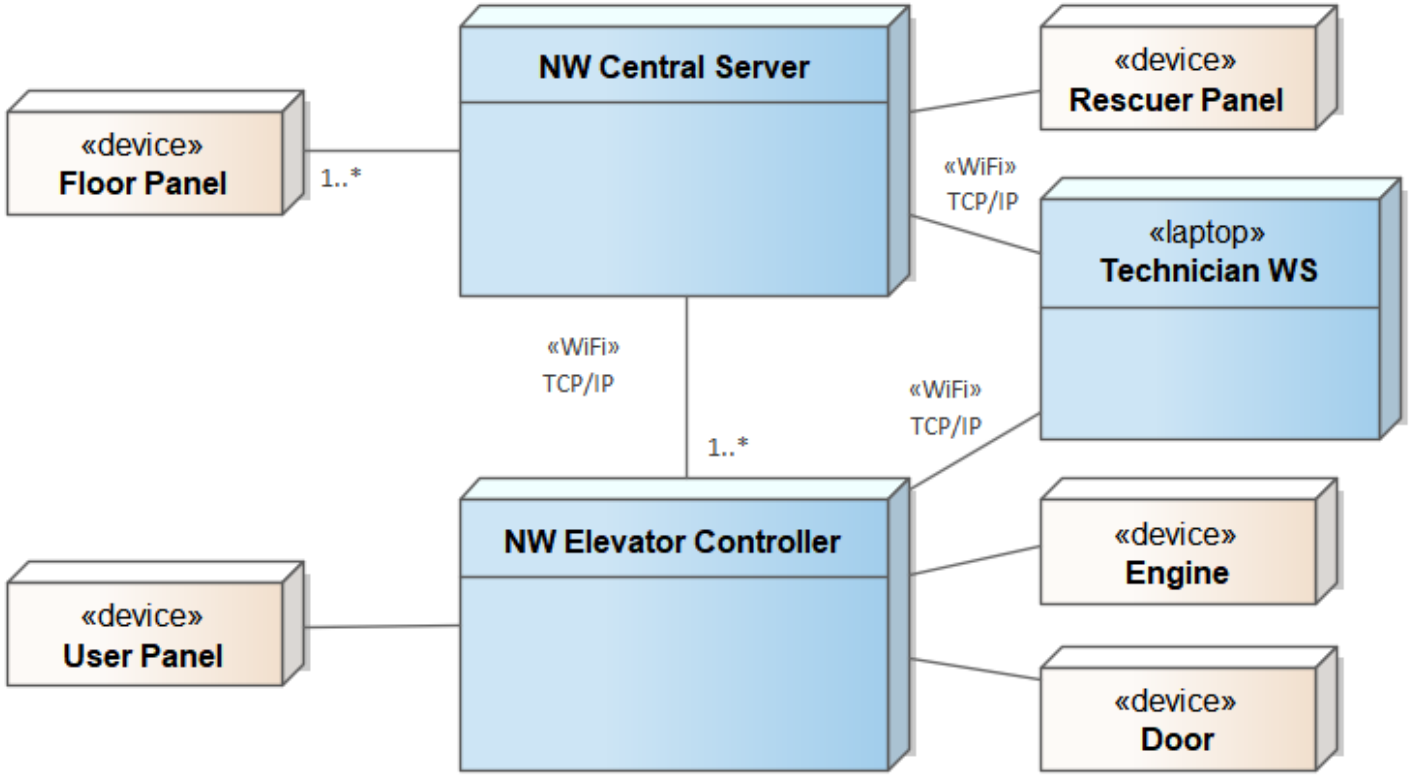
Building composite architecture: Step 3.1



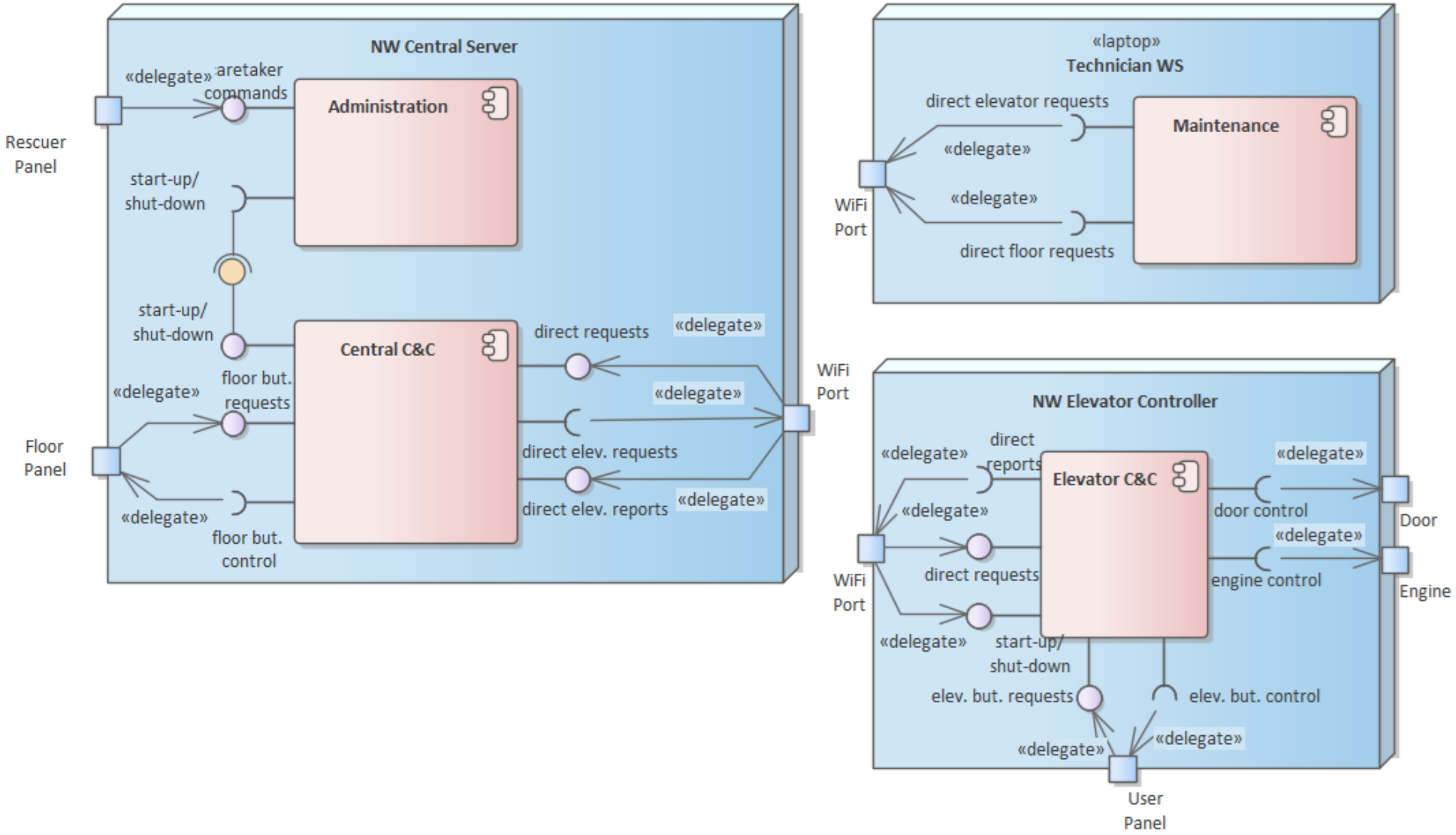
Elevator System Physical Architecture (v2)

- Network architecture
  - Central server and elevators connected via a wireless network
  - Technician comes with a laptop and connects to the system via the network

Reminder



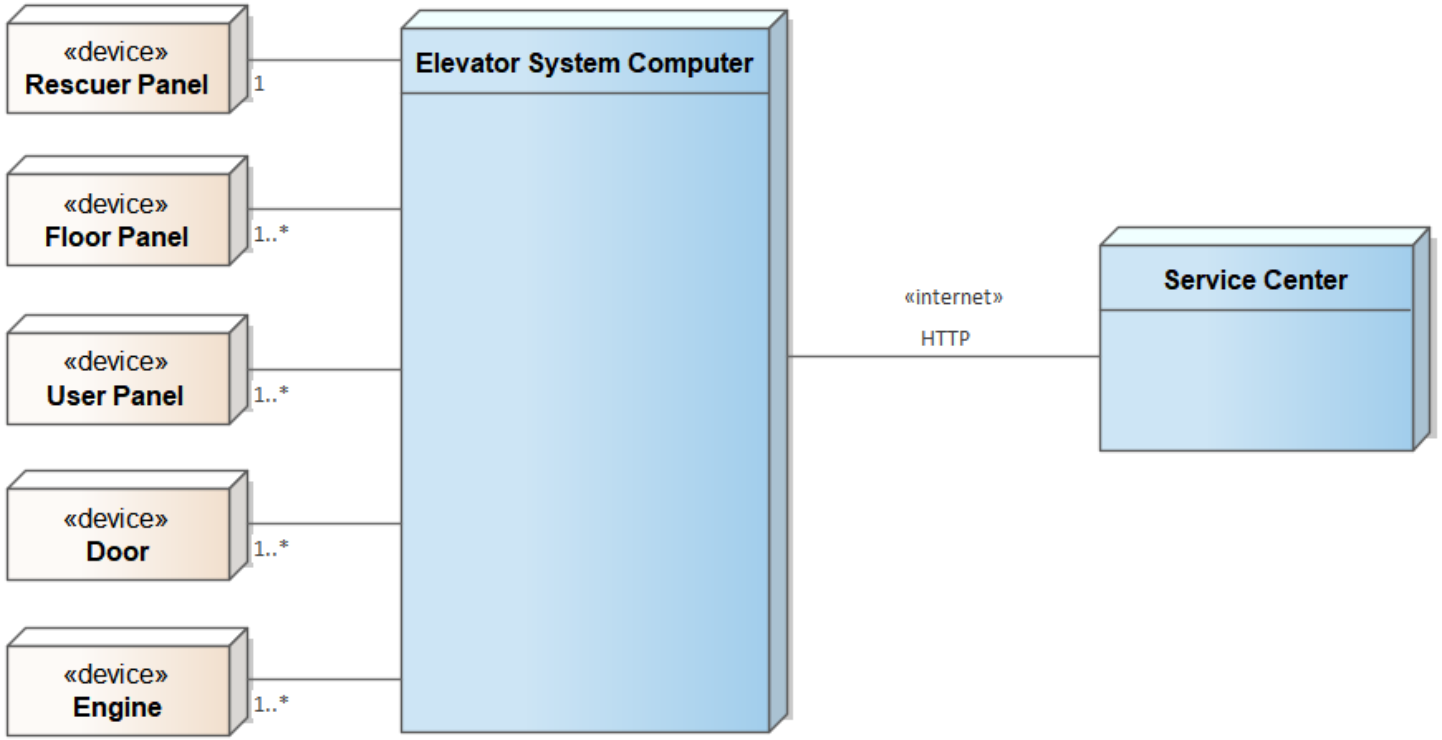
Elevator composite architecture B



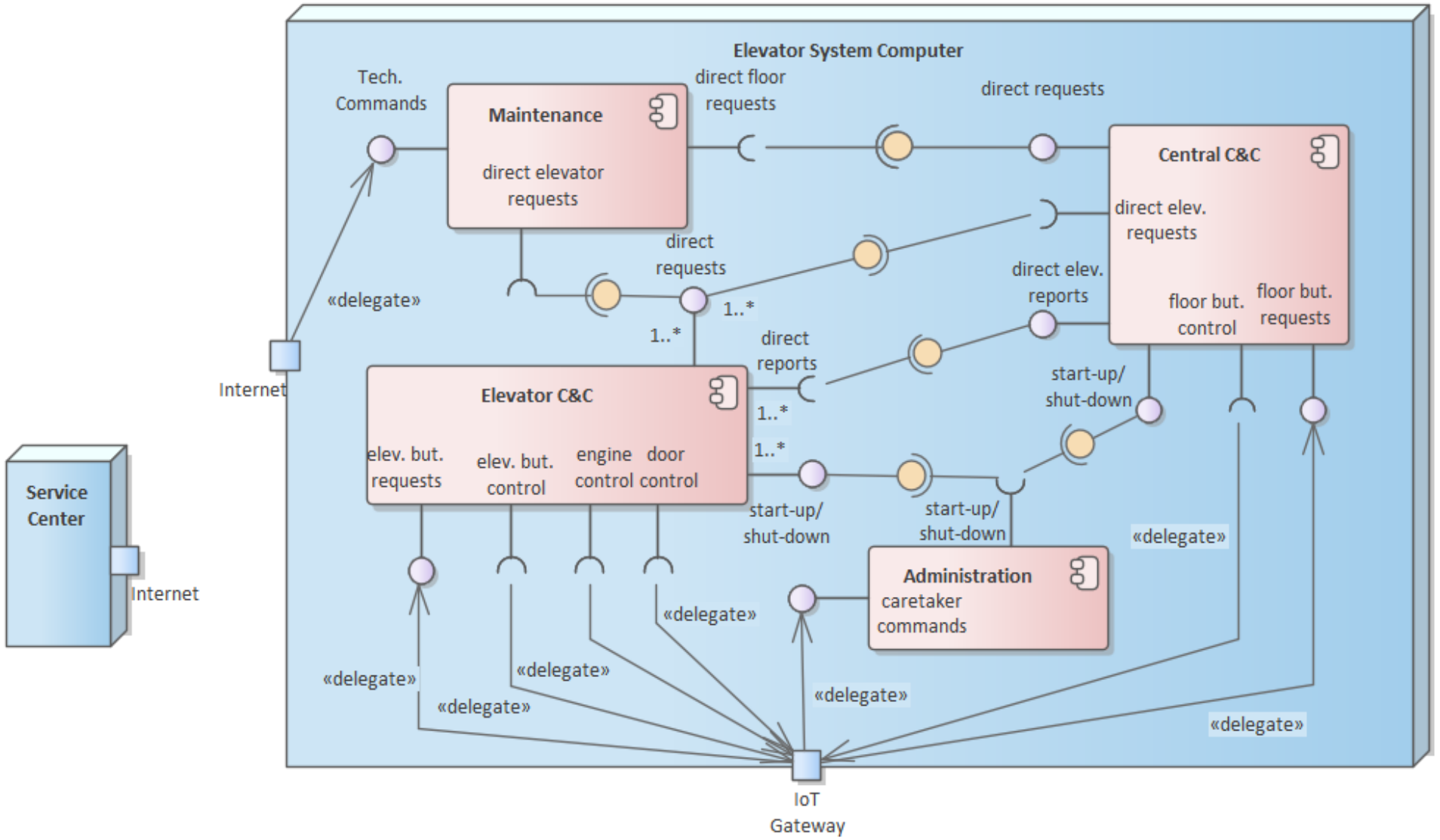
Elevator System Physical Architecture (v3)

- Centralized architecture
  - The whole system is controlled and operated by a single computer with IoT connections to all devices
  - External services (operation and control) are offered remotely via the internet

Reminder



Elevator composite architecture C



Physical architecture in UML: Software components and deployment

Artifact

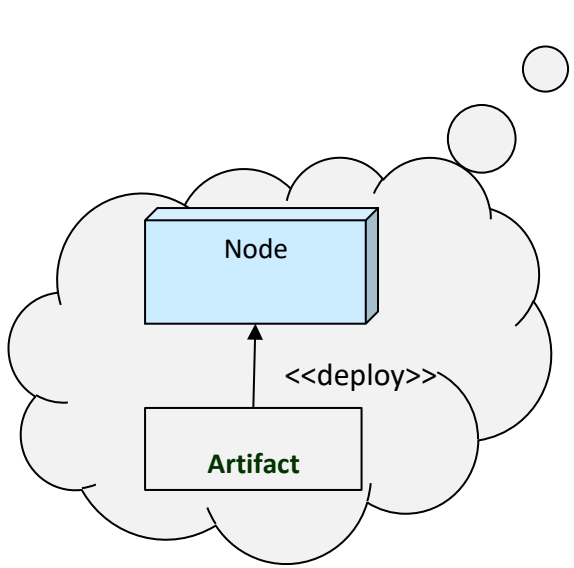
- Two kinds
- Artifacts that exist in the given computational environment (e.g., SendMsg.dll)
- Artifacts created during development (e.g., MyProg.exe)

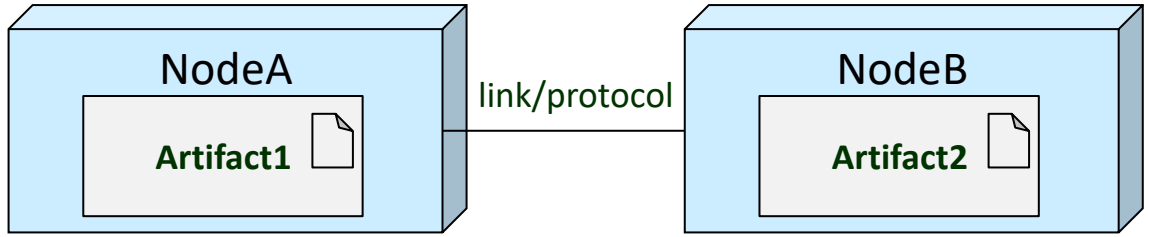
Deployment

- A dependency between a software artifact and the node it's installed on

Deployment Diagram

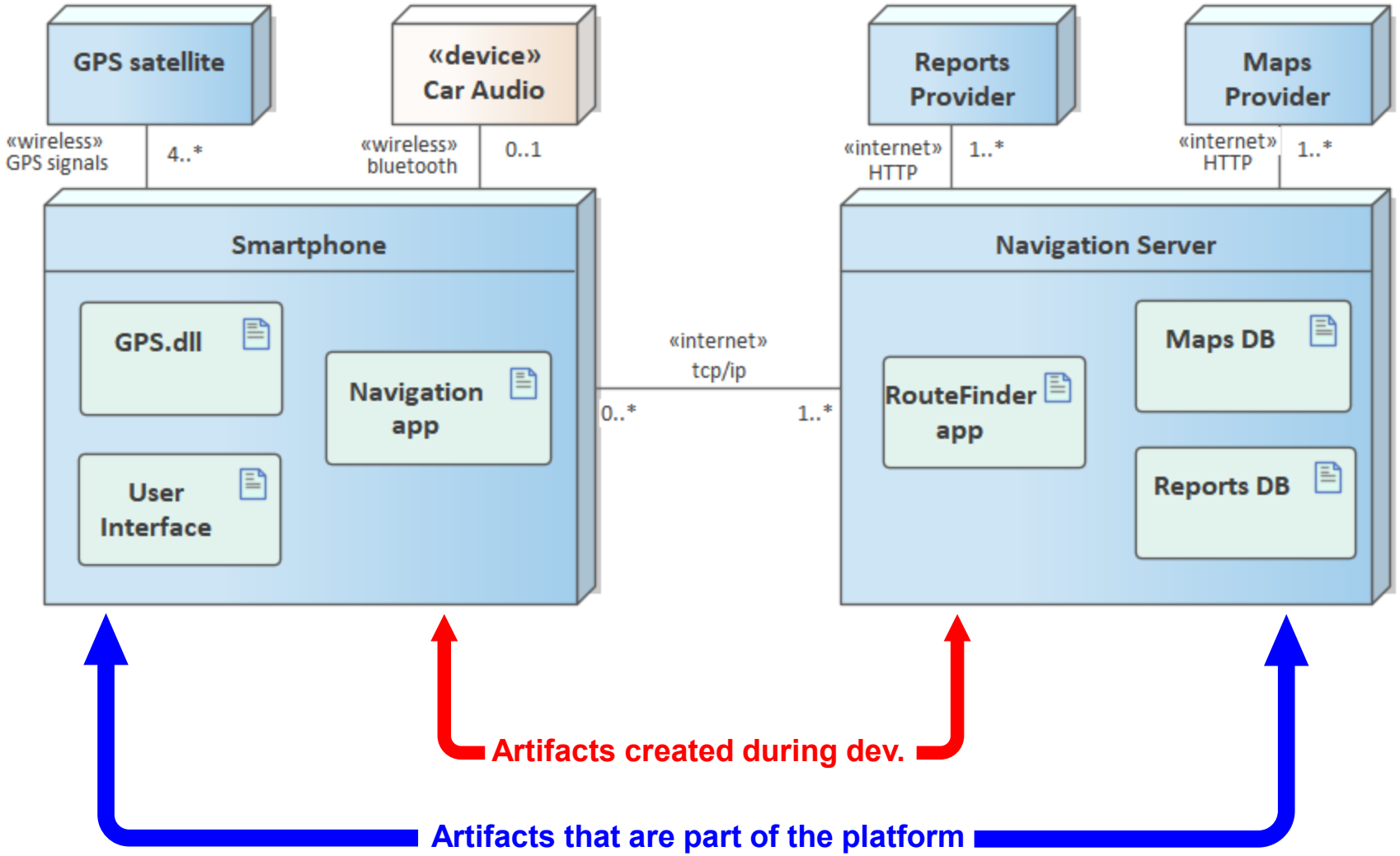
- Diagram that shows the deployment of software on hardware
- In practice, the hardware diagram showing where software is





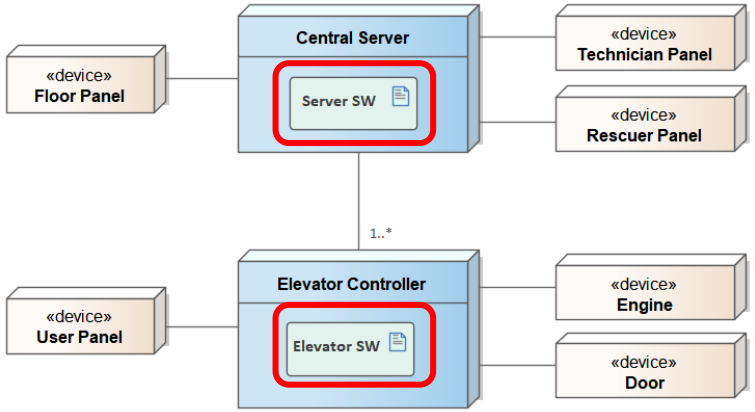
Deployment diagram with installed artifacts

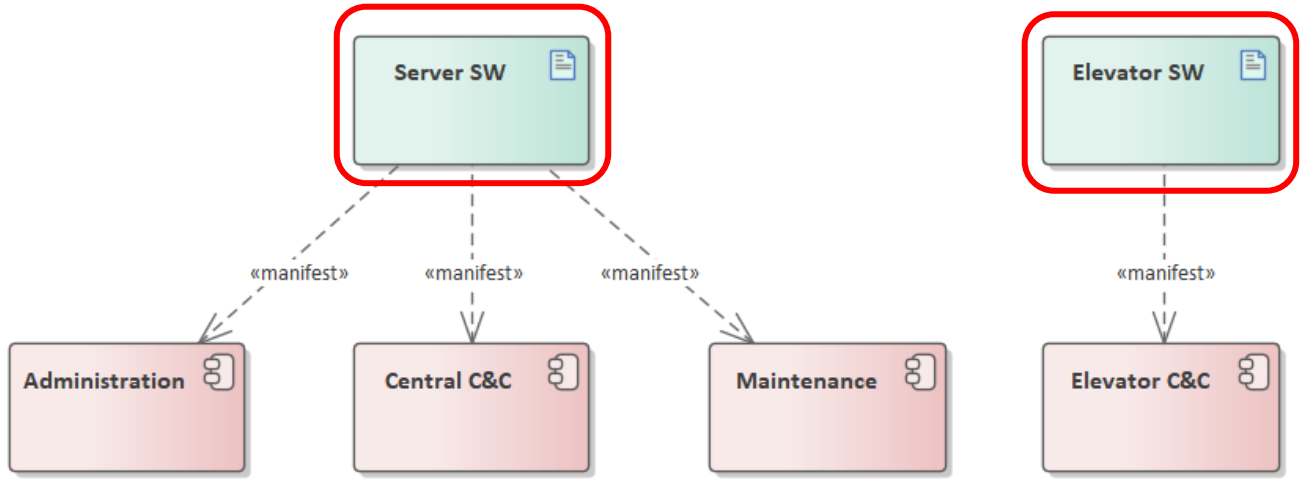
- Example: In-car navigation



Connection between artifacts and components

- Components = Development code modules
- Artifacts = installed files (.exe)
- Components are “built” into artifacts
  - Artifacts offer the components they are built of
  - The offer is reflected by the <<manifest>> dependency
- Artifacts are installed on computers in the physical architecture
  - Example elevator architecture A





In class assignment: Composite Architecture

- Create a new Composite Architecture diagram for ePark
  - Drag components and hardware nodes from your existing physical and logical architecture diagrams
  - Add ports for physical connections
  - Add “User Port” for user interfaces (GUI)
  - Connect external interfaces to ports using Delegate arrows

In class assignment: Building artifacts and installation

- Create a new Deployment Diagram called “Build Allocation”
  - Drag into it the components from the Component View
  - Based on the composite architecture, define new artifacts
  - Create <<manifest>> relationships between components and artifacts
  - Install the artifacts on the on the physical architecture

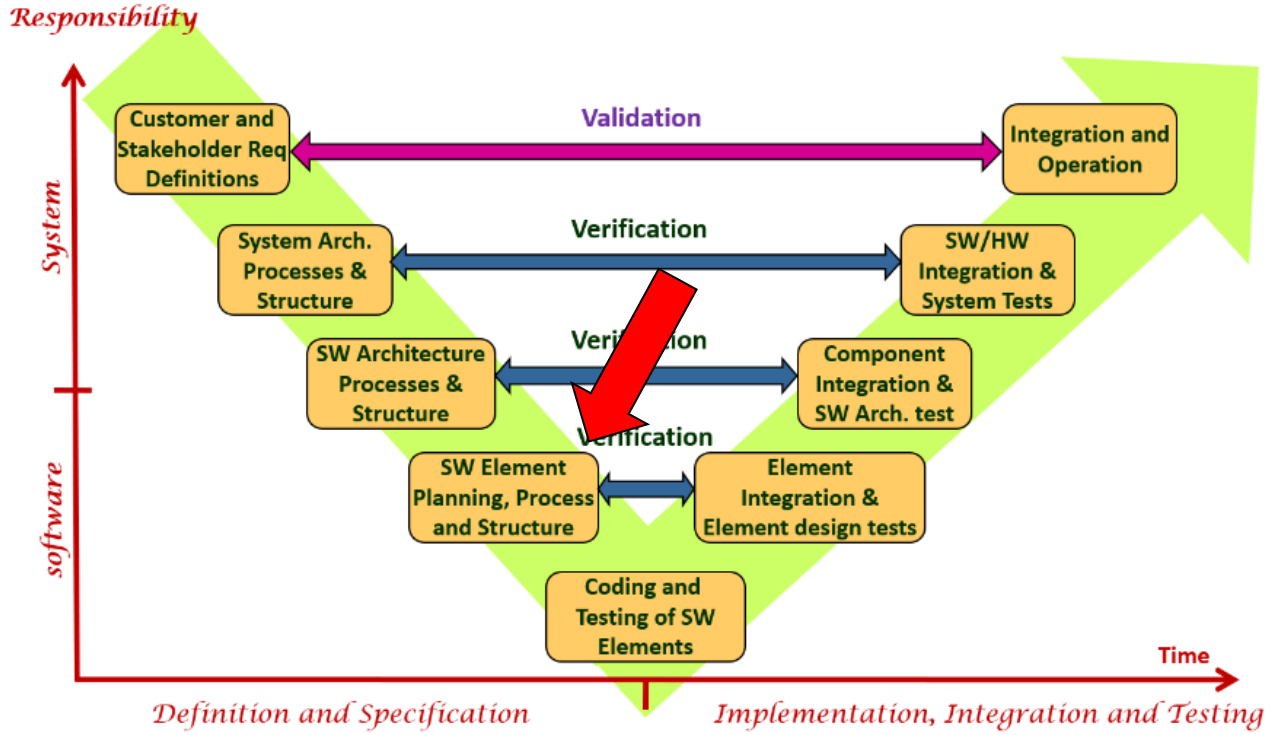
- Perform the steps above on the ePark architecture
  - Work using the steps shown in the slides before

So Far

- Composite Architecture
- PDOM

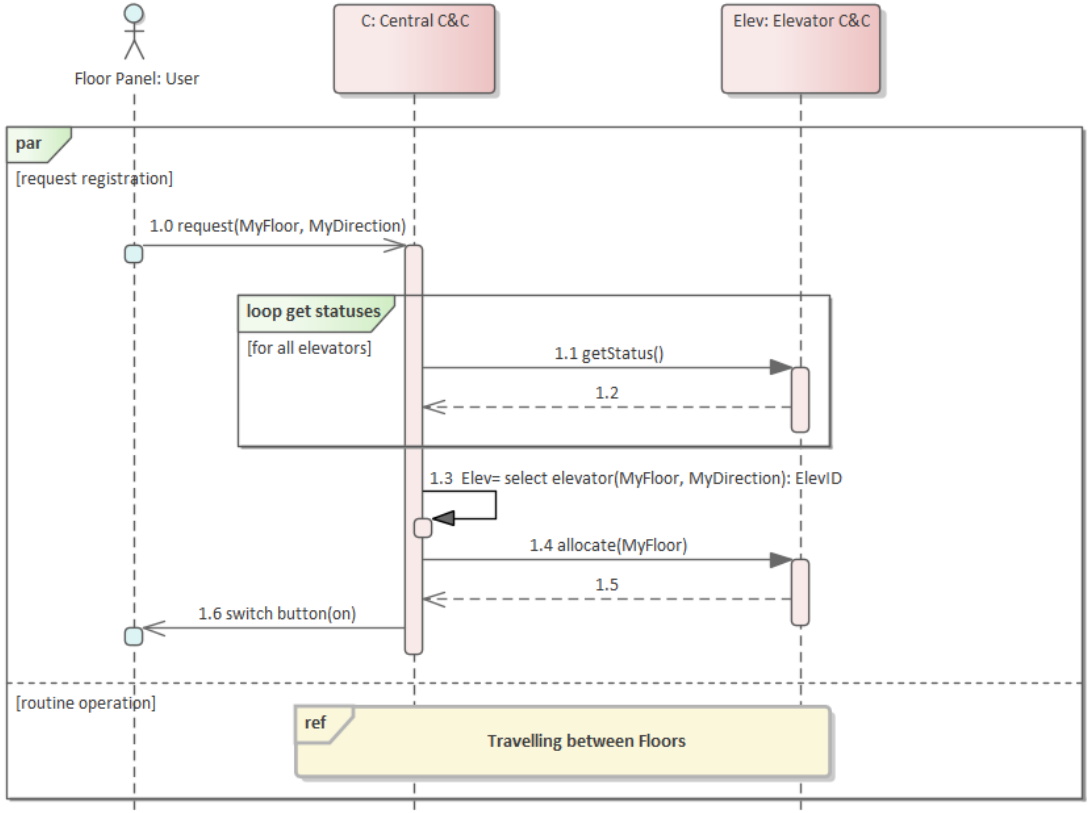
Planning software structure

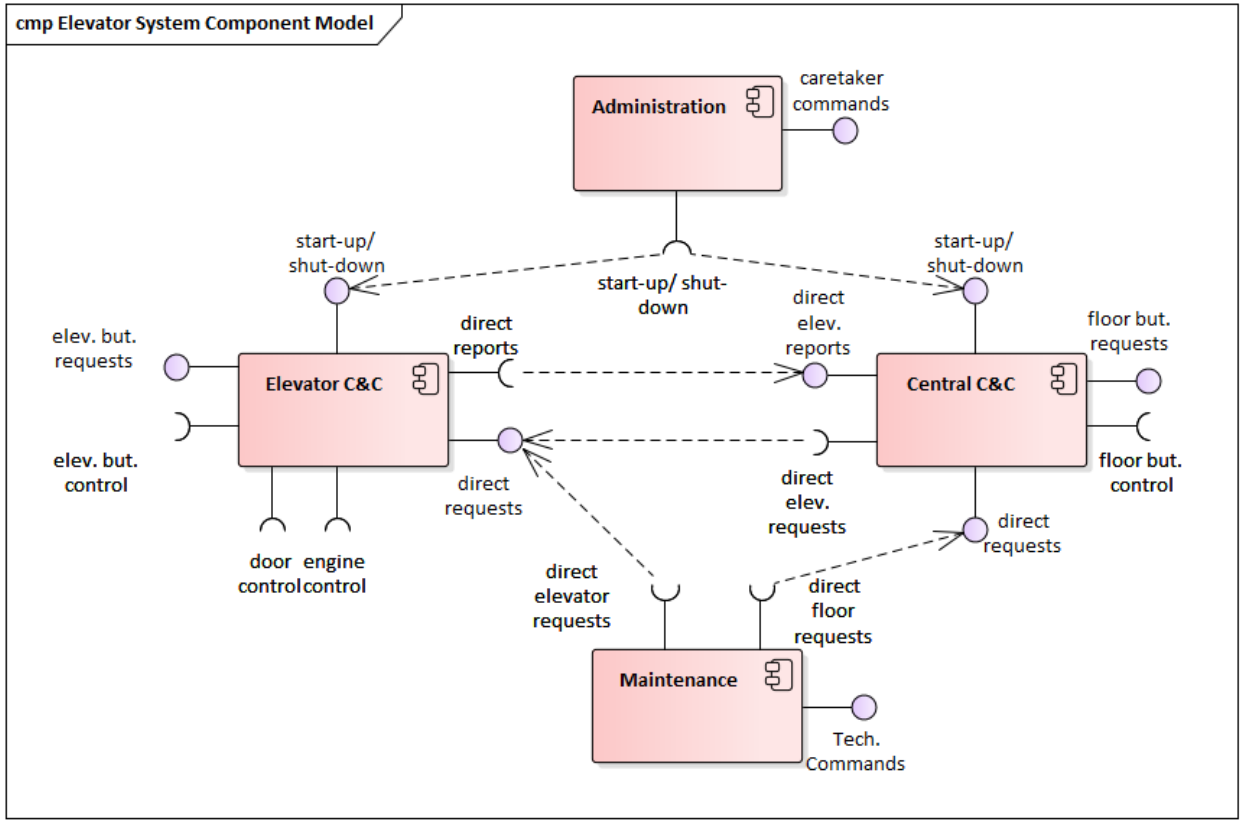
- Our goals:
  - Define the modules (classes that yield objects) that comprise the software
  - Assign functionality to classes (attributes and methods)
- Inputs:
  - Component Diagram
  - Component-level sequence diagrams
- Outputs:
  - Class diagram



What we have so far

- We built a software architecture that includes
  - **Structure**: the internal and external connections between parts – component diagram
  - **Behavior**: the interaction between components to implement tasks – sequence diagrams

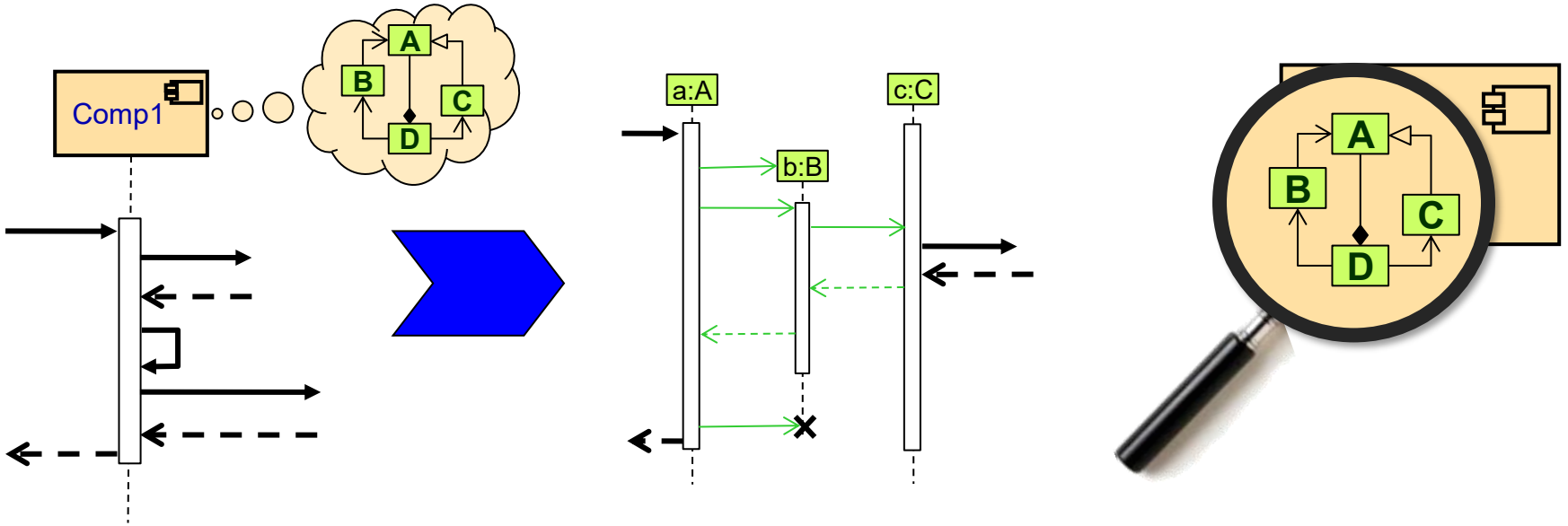




Object Oriented Design

- We need to break down each software component next

Goals	Ingredients	Structure	Behavior
• Implement each component's functionality	• Software modules (classes) that will comprise them	• The connections between various objects	• The interaction between the objects & between them and the environment that implements functionality



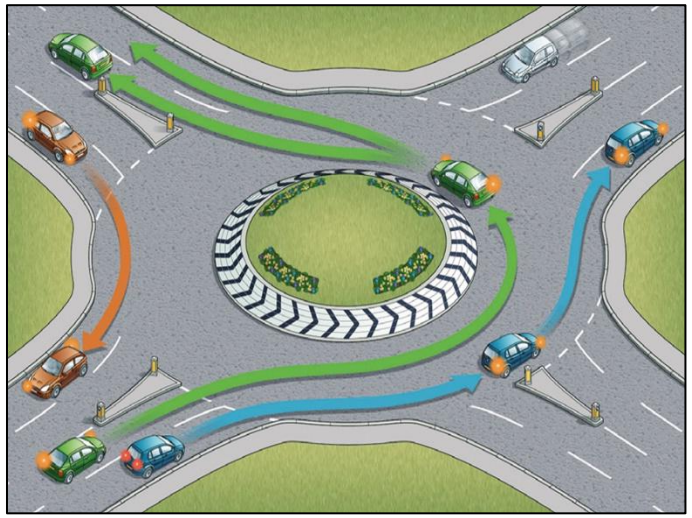
Traffic light vs. Roundabout

Traffic light intersection

- Vehicles are passive
- The algorithm is known only to the traffic light
- Traffic light manages all vehicles

Roundabout intersection

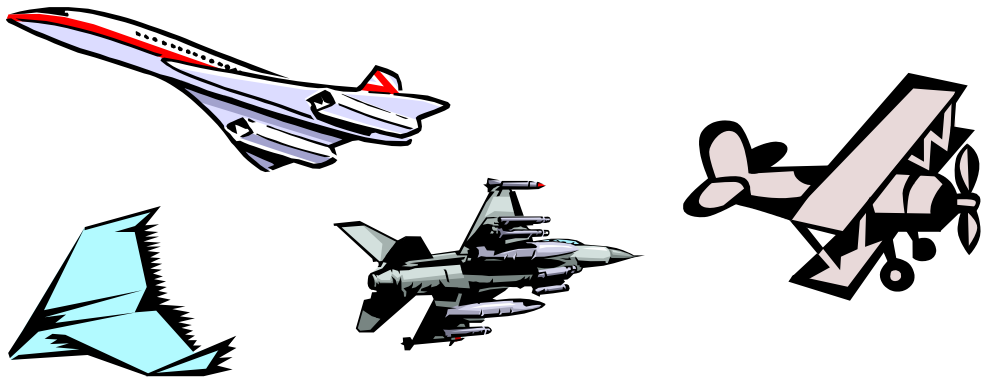
- Vehicles are active
- Each vehicle knows the algorithm
- Each vehicle manages itself and its interactions with other vehicles



Object Oriented Paradigm

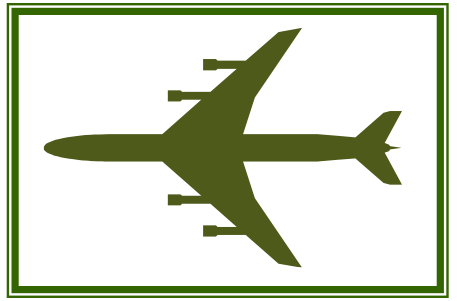
Object

- Specific entity
- Borders and identity defined
- Encapsulates state and behavior
  - State = data members, data structures
  - Behavior = member methods, functions



Class

- Descriptor of a set of objects with shared attributes
  - Properties, actions, relationships, behavior

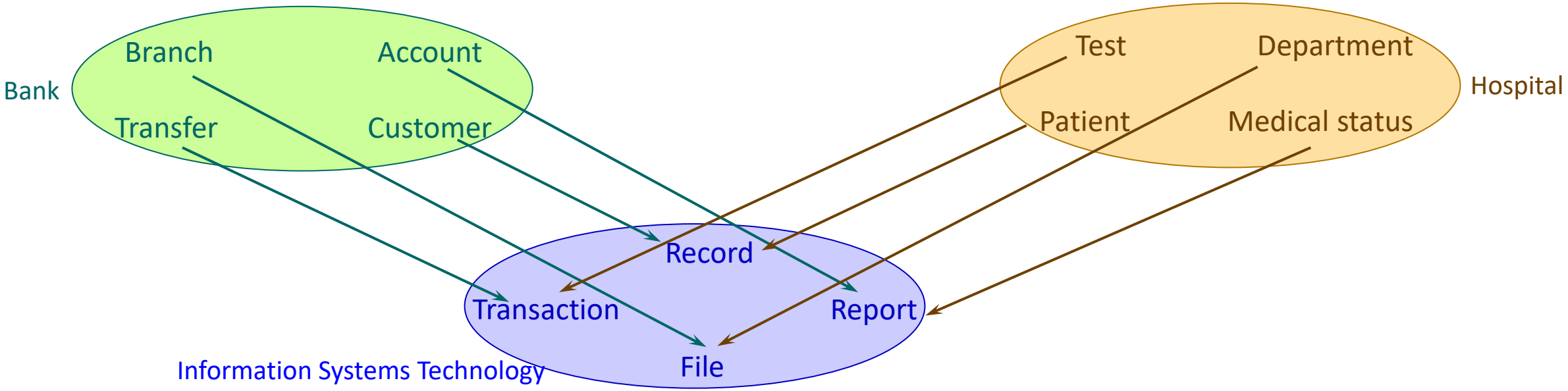


Classes exist in code only at code time
Objects exist in memory only at run time

Problem space and Solution Space

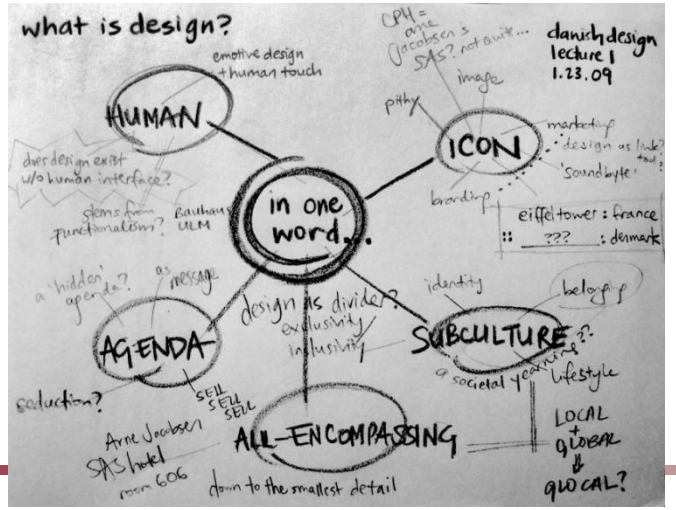
Reminder

- A development process solves a problem or a need by mapping from the problem space to the solution space
  - Solution space: Technology, engineering results
  - Problem space varies from system to system
- Example: Information Systems Technology
  - Solution space: Data records, files, reports, transactions
  - Problem space 1: A bank = {branch, customer, account, transfers}
  - Problem space 2: A hospital = {department, test, patient, diagnosis}



OO Modeling of Problem Space

- OO allows us to build software that matches the problem environment
  - We later add solution-specific elements
- During system analysis we can build a structured models of objects that reflect the problem space
  - Problem domain object model (PDOM)

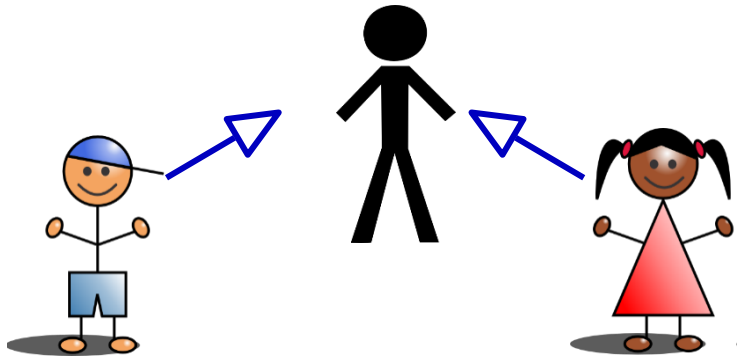


- Goal: Clarify and specify concepts and their relationships
  - Create a common language for stakeholders
- Uses:
  - Resolve contradictions and gray areas
  - System glossary
  - Basis for software objects
- Technique: Semantic network

Semantic network

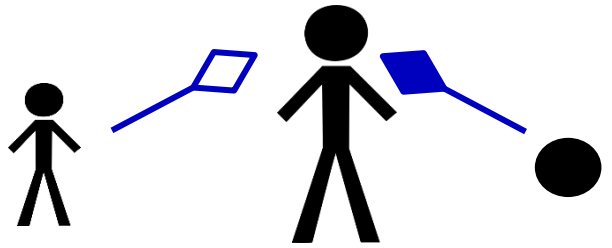
- Collection of concepts and relationships between them
- UML recognizes three kinds of relationships
  - Kind of
  - Part of/Has a
  - Relates to

- A is a B
  - A Boy is a kind of Person
  - A Girl is a kind of Person



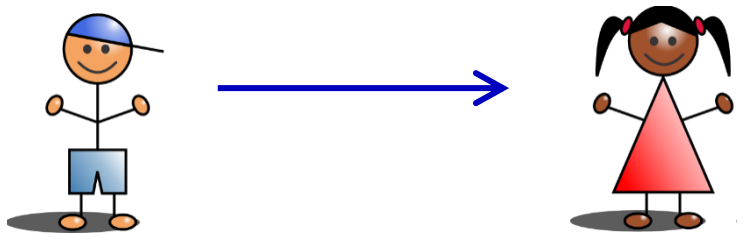
Semantic network

- Collection of concepts and relationships between them
- UML recognizes three kinds of relationships
  - Kind of
  - Part of/Has a
  - Relates to
- A has a B
  - A Head is part of a Person
  - A Person has a Child



Semantic network

- Collection of concepts and relationships between them
- UML recognizes three kinds of relationships
  - Kind of
  - Part of/Has a
  - Relates to
- A <relates to> B
  - A Boy works with a Girl



Operational Specification: Elevator System

Reminder

A passenger who is on a particular floor and wants to call an elevator presses the appropriate button for the direction of travel (up or down). If the button was not already lit, the button lights up after being pressed. An elevator car traveling in the requested direction will arrive at the floor within one minute at most. When the car arrives, the door opens and the light on the button turns off.

A passenger who is in an elevator car and wants to travel to a particular floor presses on the button associated with the desired floor. If the button wasn't already lit, the button lights up and a new stop request for the floor is registered. The door closes. After a short pause, the car continues moving. It stops at every floor for which there is a stop request. When the car stops at a floor, the door opens and the button for the floor turns off.

A passenger can stop the elevator car when its moving by pressing on the emergency stop button. In that case, the elevator stops immediately and all registered stop requests are erased. Afterwards, the elevator can resume operation by pressing a button for any floor.

Operational Specification: Elevator System

Reminder

If the elevator gets stuck while in operation, passengers can call for help using the emergency rescue button. The rescuer (the building maintenance engineer) will access the emergency panel in the machine room and perform operations to move the elevator car to the bottom floor and open the door.

The maintenance engineer is responsible to start up the elevators at the beginning of the day and to shut it down at the end of the day. The technician comes once every 6 months and performs a complete check of the system and fixes problems using the technician panel in the machine room.

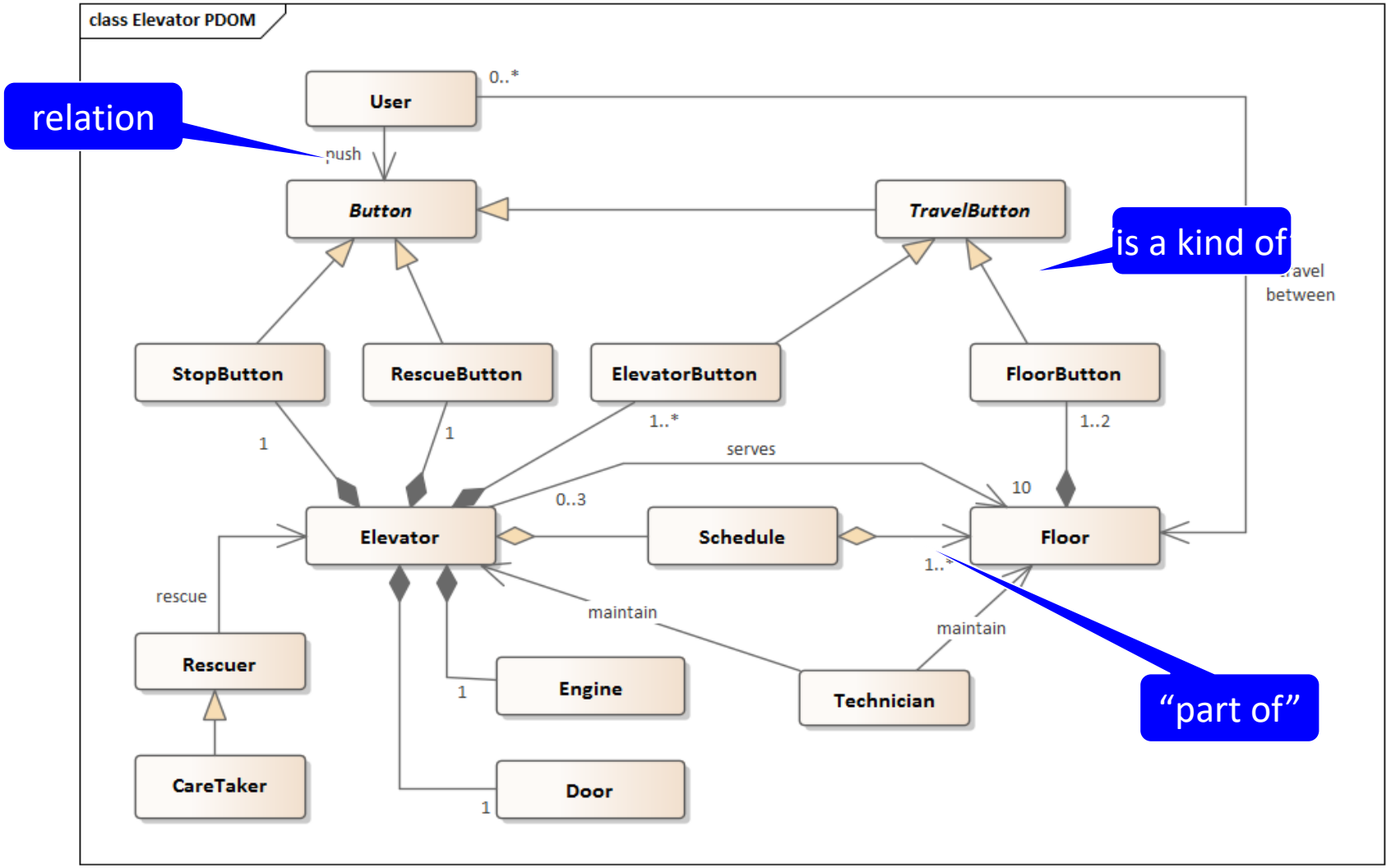
The elevator system must meet all applicable safety rules.

The system must be accessible to the handicapped.

List of Elevator Concepts

- Passenger / User
- Floor
- Elevator
- Directional button (floor)
- Door
- Engine
- Floor Travel button (in car)
- Stop request
- Emergency stop
- Rescue panel
- Caretaker/Maintenance engineer
- Check/Test
- Problems/faults
- Technician
- Travel Schedule

Elevator PDOM



In class assignment: PDOM

- Create a PDOM for the ePark system
  - Use the Class Diagram Toolbox in Enterprise Architect
  - Include the following concepts (you can add your own if you need)

1. Child's Guardian
2. Account
3. Child
4. eTicket
5. Bracelet
6. Entry
7. Attraction (device)
8. Supervisor

Conclusion

- Composite Architecture
- PDOM