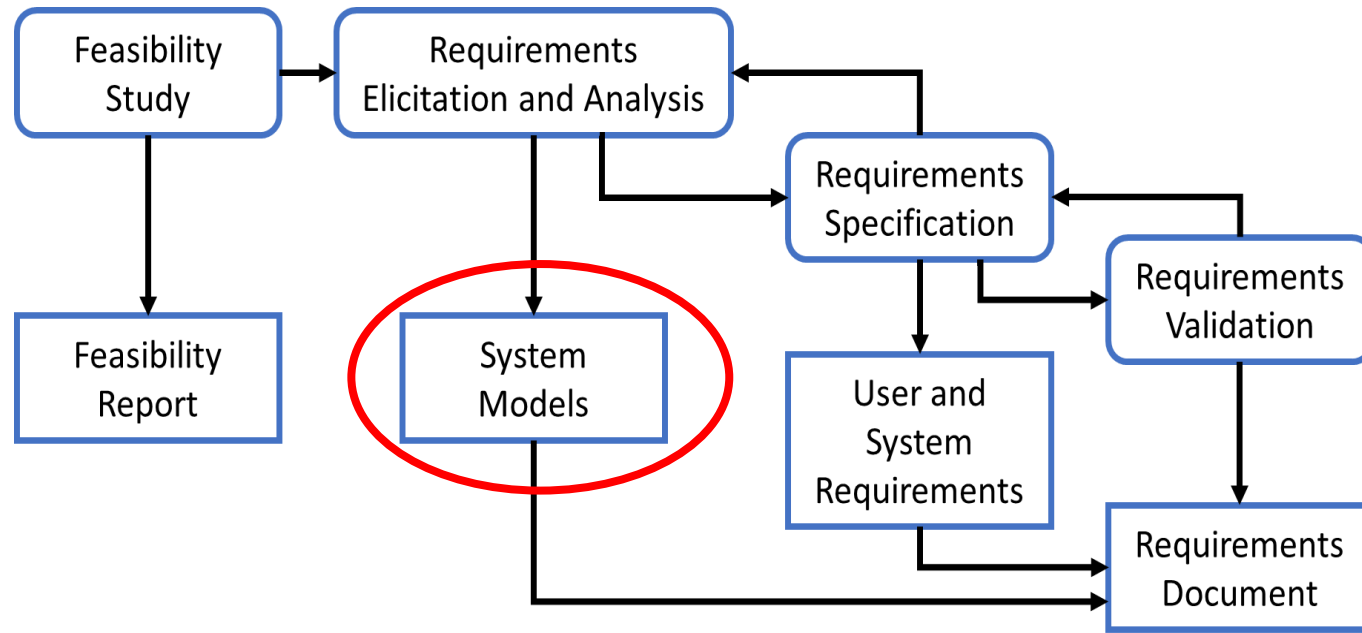


# Software Engineering COMP1035

## Lecture 05

### *Requirements Modelling*





2

---

# System Modelling

# What Are Requirements Models For?

---



# Requirements Models

---

- Models are utilised throughout the requirements engineering process to **facilitate the derivation of detailed system requirements**.
- They serve to **describe the system for engineers involved** in its implementation and, post-implementation, to **record the system's structure and functionality**.
- Models can be designed for:
  - For existing systems, models aid in requirements engineering by **clarifying the functionality of the current system**, thereby directing **stakeholder discussions** towards its merits and drawbacks.
  - For new systems, models are used during requirements engineering in **clarifying the proposed requirements to various system stakeholders**. Engineers employ these models to discuss on design proposals and to document the system for implementation.

# Requirements Models

---

“It is **not a complete** representation of the system.”

“It purposely leaves out details and picks out **the most salient** characteristics.”

*- Ian Sommerville (2011) Chapter 5, Systems Modelling*

- **Synthesising** all the requirements you collected into key requirements.
- You start making diagrams that represent how **requirements relate**.
- Then you have a **comprehensive set** of **integrated requirements**.

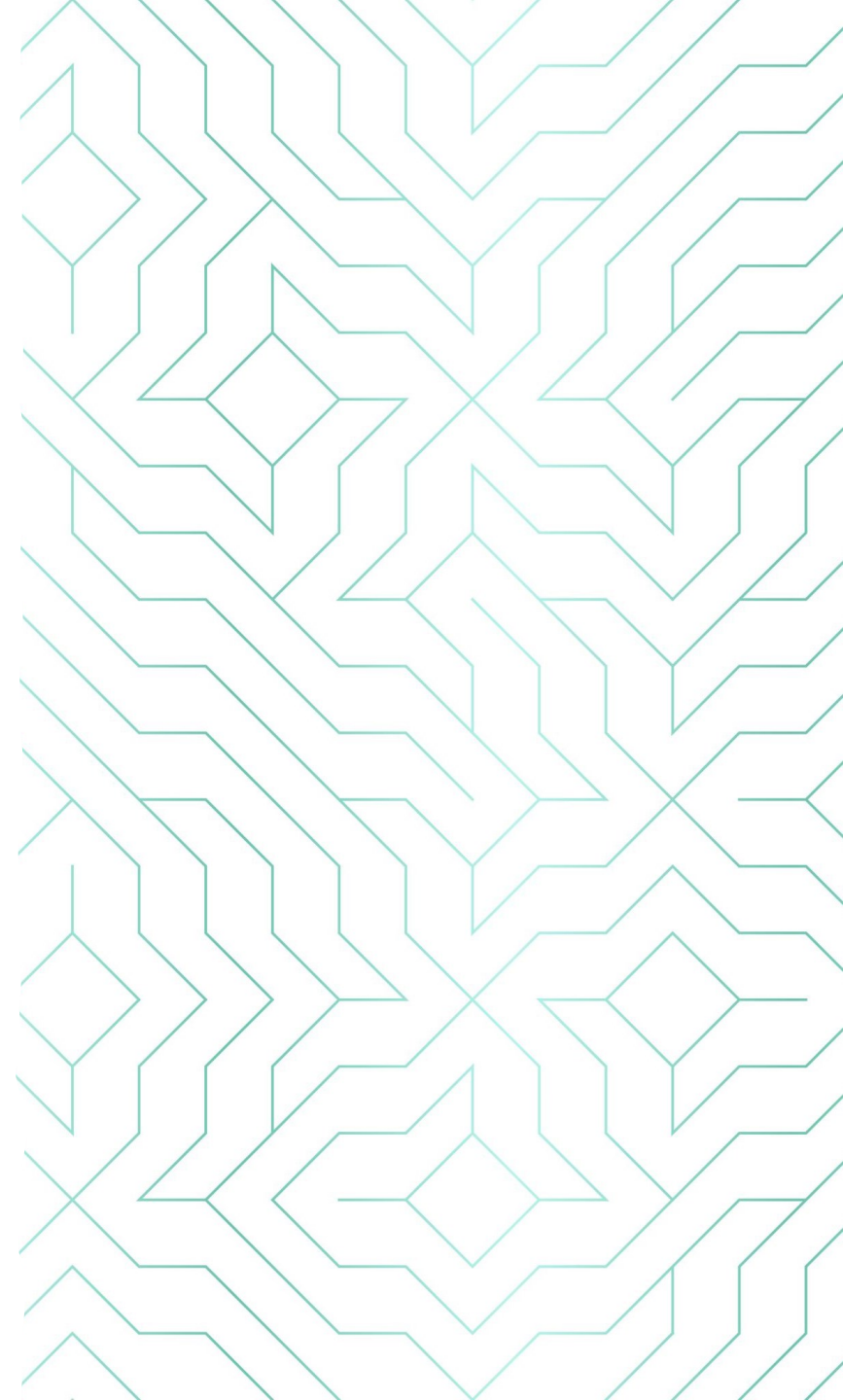
# Requirements Models

- Different models are used to represent system from **different perspectives**:
  - **External**: Model the context or environment of the system.
  - **Interaction**: Model the interactions between a system and its environment, or between the components of a system.
  - **Structural**: Model the organization of a system or the structure or the data processed by the system.
  - **Behavioral**: Model the dynamic behavior of the system and how it responds to events.

# UML Diagrams

---

Graphical notation based on diagram types for representing a system



# Unified Modeling Language (UML)

---

- There used to be three popular approaches to producing models.
  - Ones specifically for modelling object-oriented software in software engineering.
- In 1990, they got “unified”.
- UML 2.5 includes 13 different diagrams (released in June 2015).
  - Each for different purposes.
- It’s possible to automatically produce outline code from UML.
  - Called Model-Driven Development.
  - But this isn’t entirely popular – the book outlines 4 reasons why in section 5.5.

<https://www.uml-diagrams.org/uml-25-diagrams.html>



# FIVE Diagrams (Could Representing a System)

---

- **Activity Diagrams**: Show the activities involved in a process or in data processing.
- **Use Case Diagrams**: Show the interactions between a system and its environment.
- **Sequence Diagrams**: Show interactions between actors and the system and between system components.
- **Class Diagrams**: Show the object classes in the system and the associations between these classes.
- **State Diagrams**: Show how the system reacts to internal and external events.

# UML – Key Diagram Types

Context Models

Activity Diagrams



Initial Requirements

Could be Req., Spec. or Doc.

Use Case Diagrams

\*Sequence Diagrams



Refined Requirements

Could be Req., Spec. or Doc.

Class Diagrams

\*Sequence Diagrams

State Diagrams



Architecture Design

Of Code – thu more for Spec. and Doc.

# Context Models

---

- During requirements gathering, you identified several (***internal or external***) systems you will have to interact with
  - E.g., authentication systems, Bluecastle, finance etc.
- You want to show how all all these systems inter-relate.
  - Use Case Diagram could have an “actor” – but that’s not what UCDs are “for”.
  - You could use a flow diagram (Activity Diagram – see later) – but that’s not what they are “for”.
- Context Models are especially **for this problem**.

# Context Diagram (Read as)

- You can see that the Mentcare system is connected to an appointments system and a more general patient record system with which it shares data. The system is also connected to systems for management reporting and hospital admissions, and a statistics system that collects information for research. Finally, it makes use of a prescription system to generate prescriptions for patients' medication.

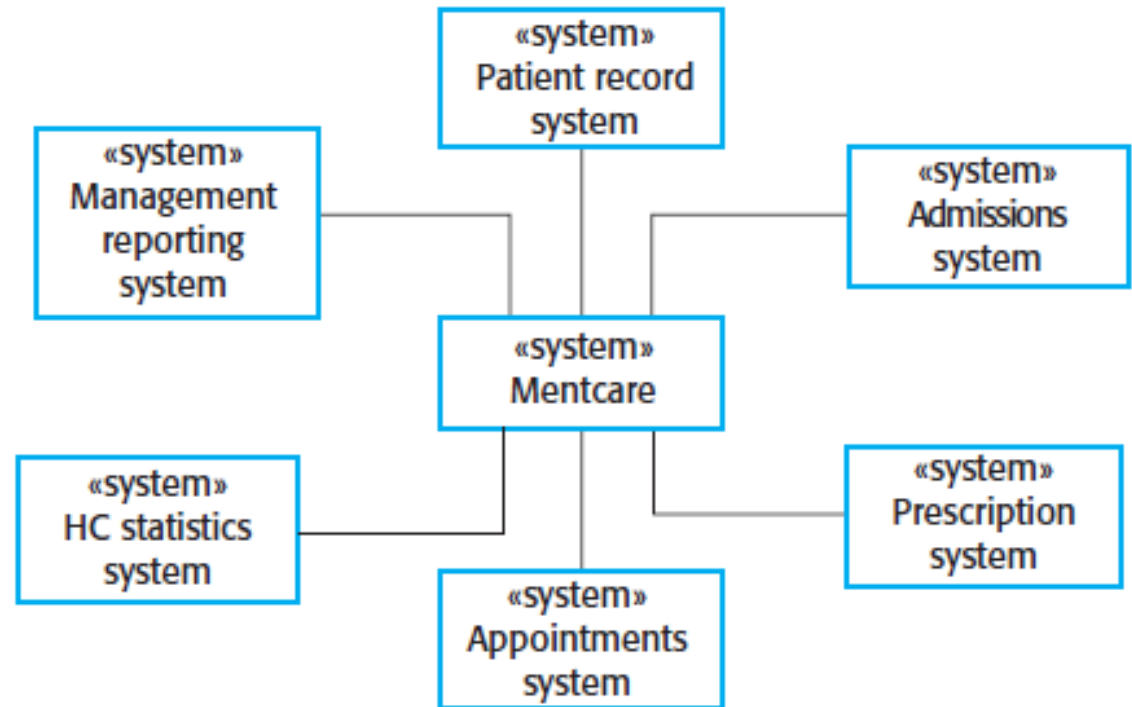


Fig: A Context Model of the Mentcare System

# Context Diagram

---

- Show related systems.
- These are often part of the Non-Functional (NF) requirements.
- Can be simple like this one.
- Even show components of a multi-part system (Twitter).

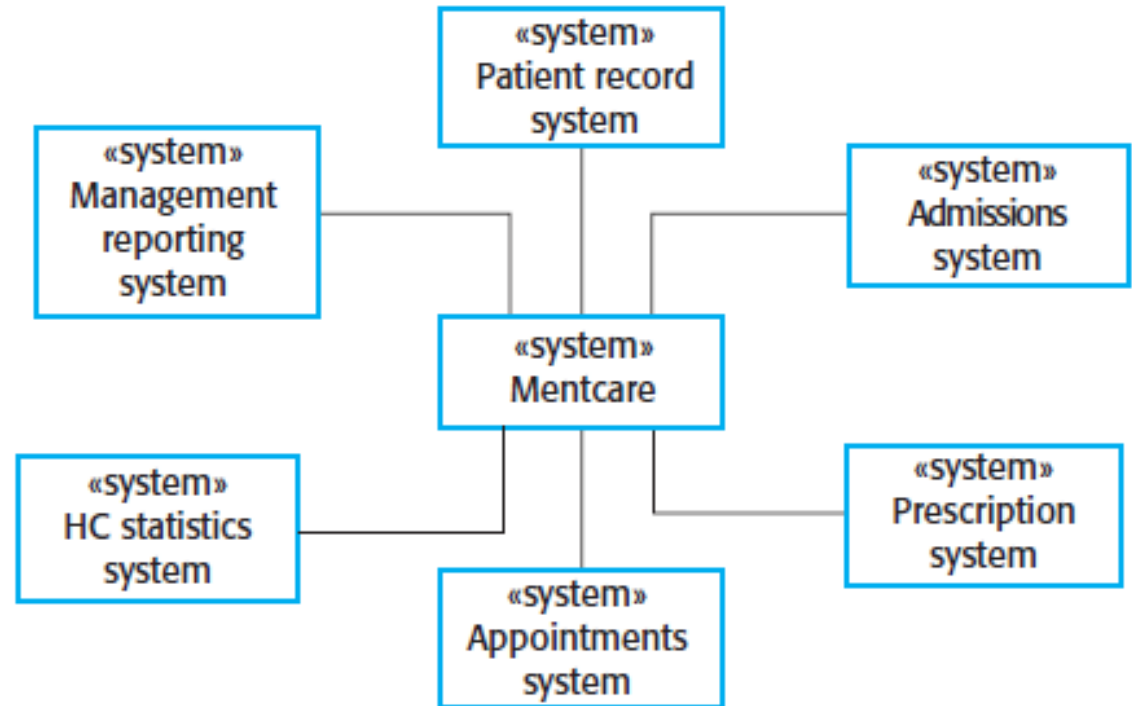


Fig: A context model of the Mentcare system



# Context Diagram

- Context models define the **boundaries** of they system.
- They represent key systems that **need to be** developed.
- And **the relationship** to the other systems/components.
- Also – **what NOT to develop** (to focus investment).

# UML – Key Diagram Types

Context Models  
Activity Diagrams



Initial Requirements

Could be Req., Spec. or Doc.

Use Case Diagrams  
\*Sequence Diagrams



Refined Requirements

Could be Req., Spec. or Doc.

Class Diagrams  
\*Sequence Diagrams  
State Diagrams



Architecture Design

Of Code – thu more for Spec. and Doc.

# Activity Diagrams

- Used to elaborate workflows for key activities – especially if they involve decisions.
- Explains the process, decision points, wait points & parallel work.
- To define **ONE** Use Case in more detail.
- May tie together several Use Cases for one bigger process.

**\*In general, ONE activity diagram per use case!**

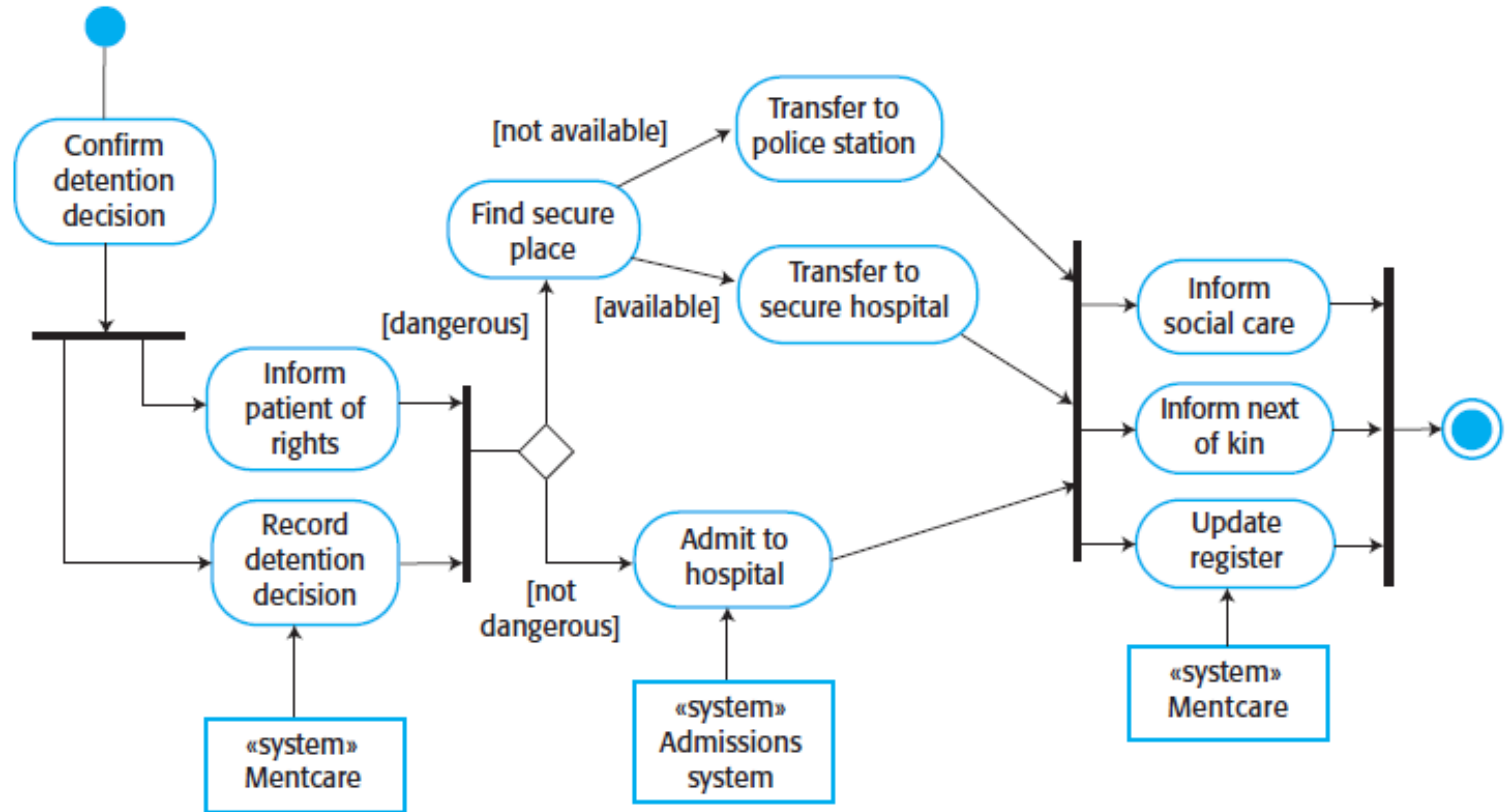


Fig: A Process Model of Involuntary Detention



# Activity Diagrams

- **Reasoning** of the diagram:
  - E.g., the decision to detain a patient must be regularly reviewed so that people are not held indefinitely without good reason. One critical function of the Mentcare system is to ensure that such safeguards are implemented and that the rights of patients are respected

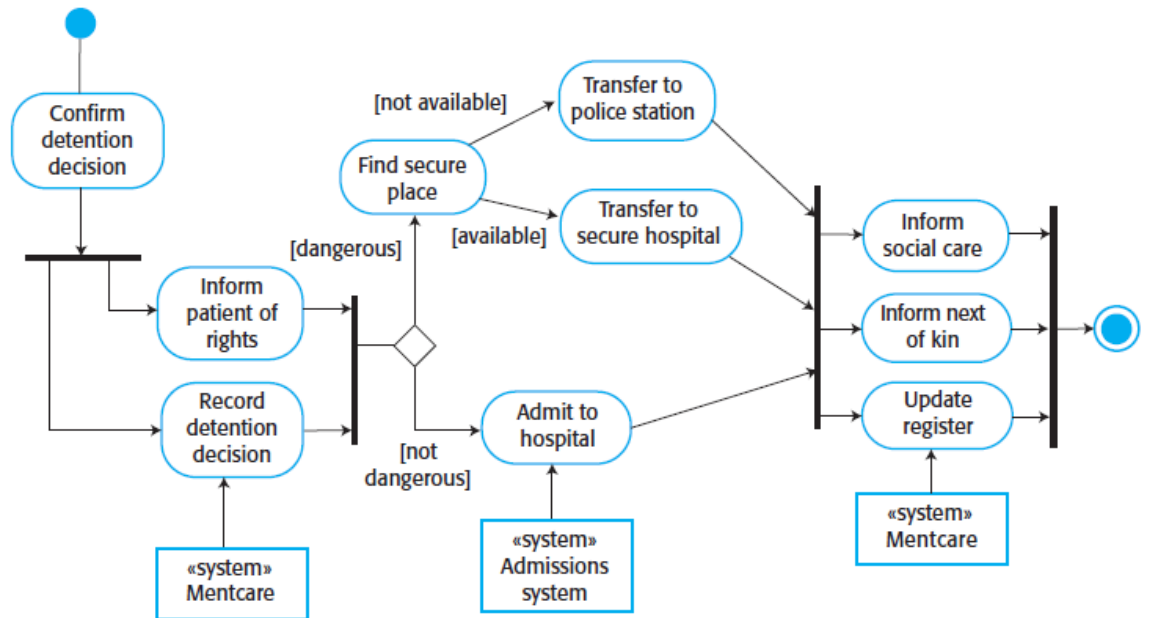
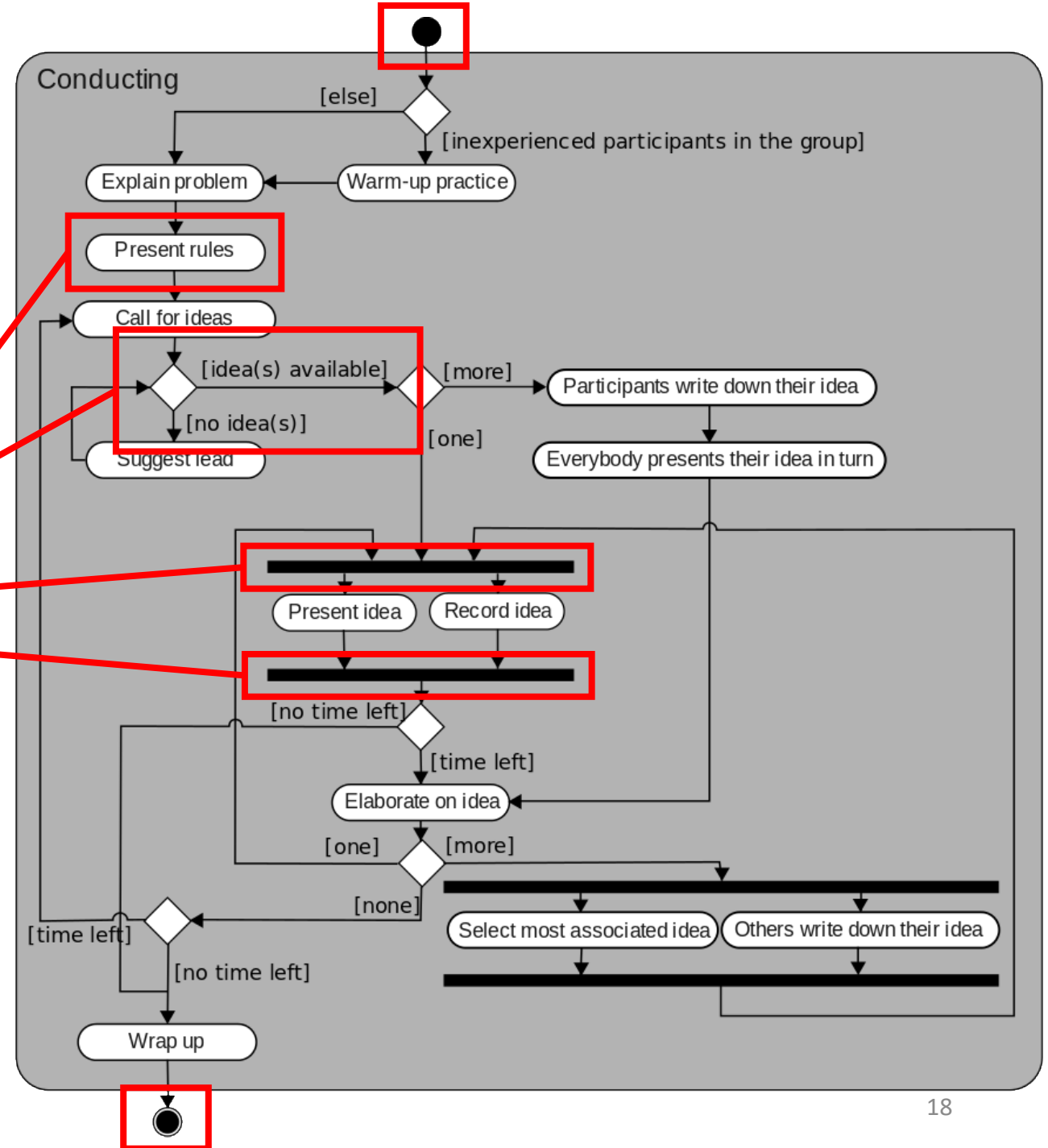


Fig: A Process Model of Involuntary Detention

# Activity Diagrams

- Rounded rectangles represent **actions**.
- Diamonds represent **decisions**.
- Bars represent the **start** (split) or **end** (join) of concurrent activities.
- The black circle represents the **start**.
- The encircled black circle represent the **end**.

[https://en.wikipedia.org/wiki/Activity\\_diagram](https://en.wikipedia.org/wiki/Activity_diagram)



# UML – Key Diagram Types

Context Models  
Activity Diagrams



Initial Requirements

Could be Req., Spec. or Doc.

Use Case Diagrams  
\*Sequence Diagrams



Refined Requirements

Could be Req., Spec. or Doc.

Class Diagrams  
\*Sequence Diagrams  
State Diagrams



Architecture Design

Of Code – thu more for Spec. and Doc.

# Use Case Diagrams & Sequence Diagrams

- Good for **complex sharing of information between people and systems.**
- Could be seen as a series of messages between key components.
- Can be (later) used to specify function calls in Java objects.

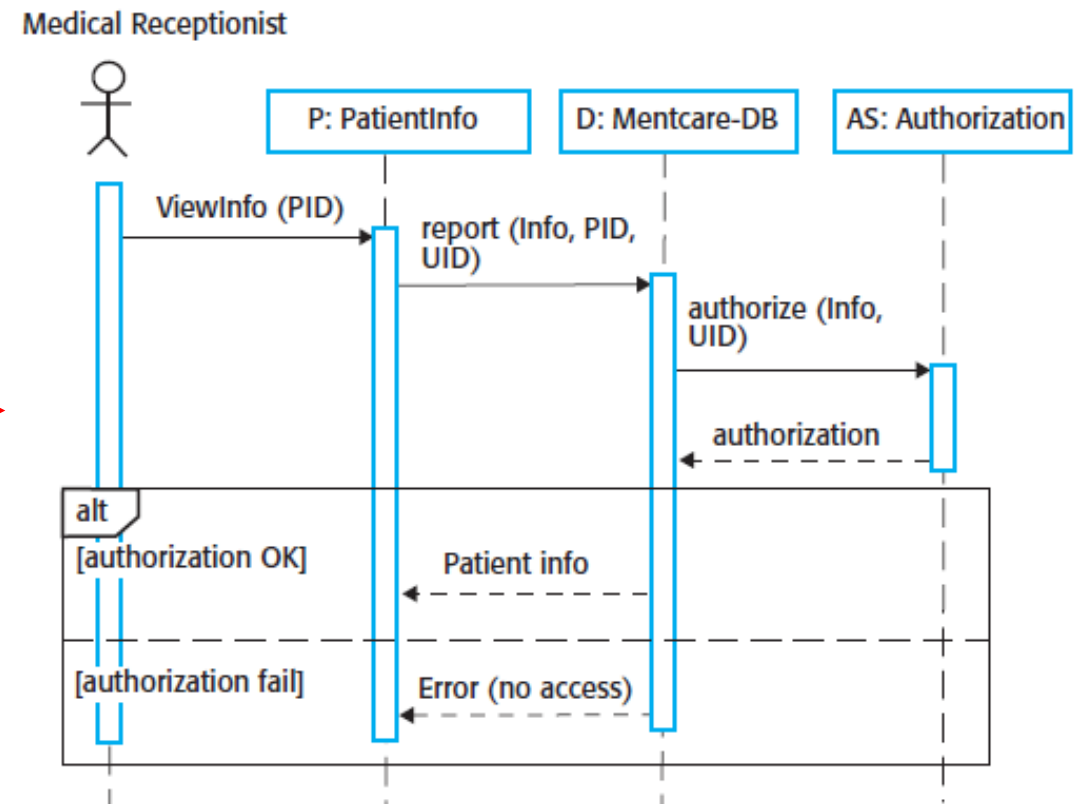
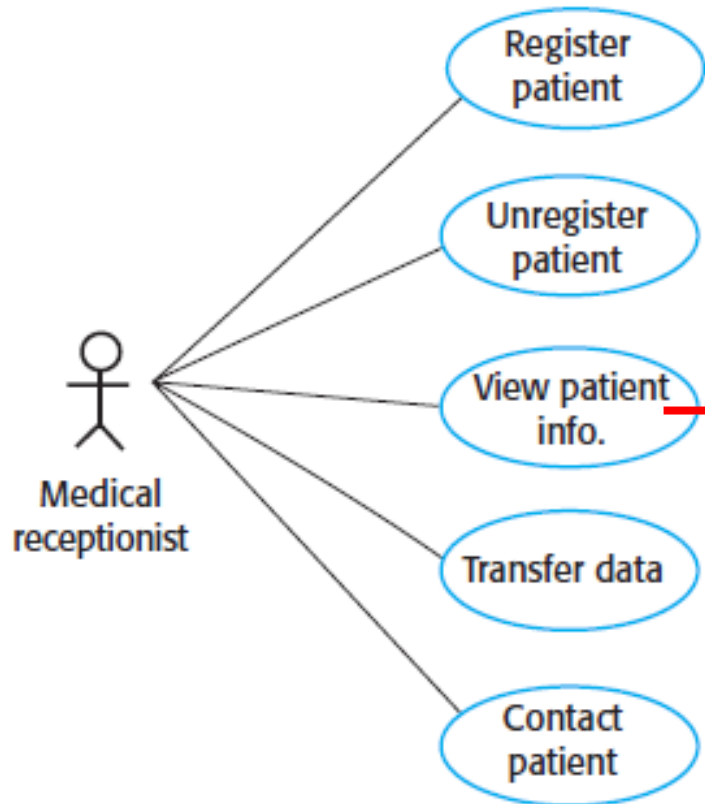
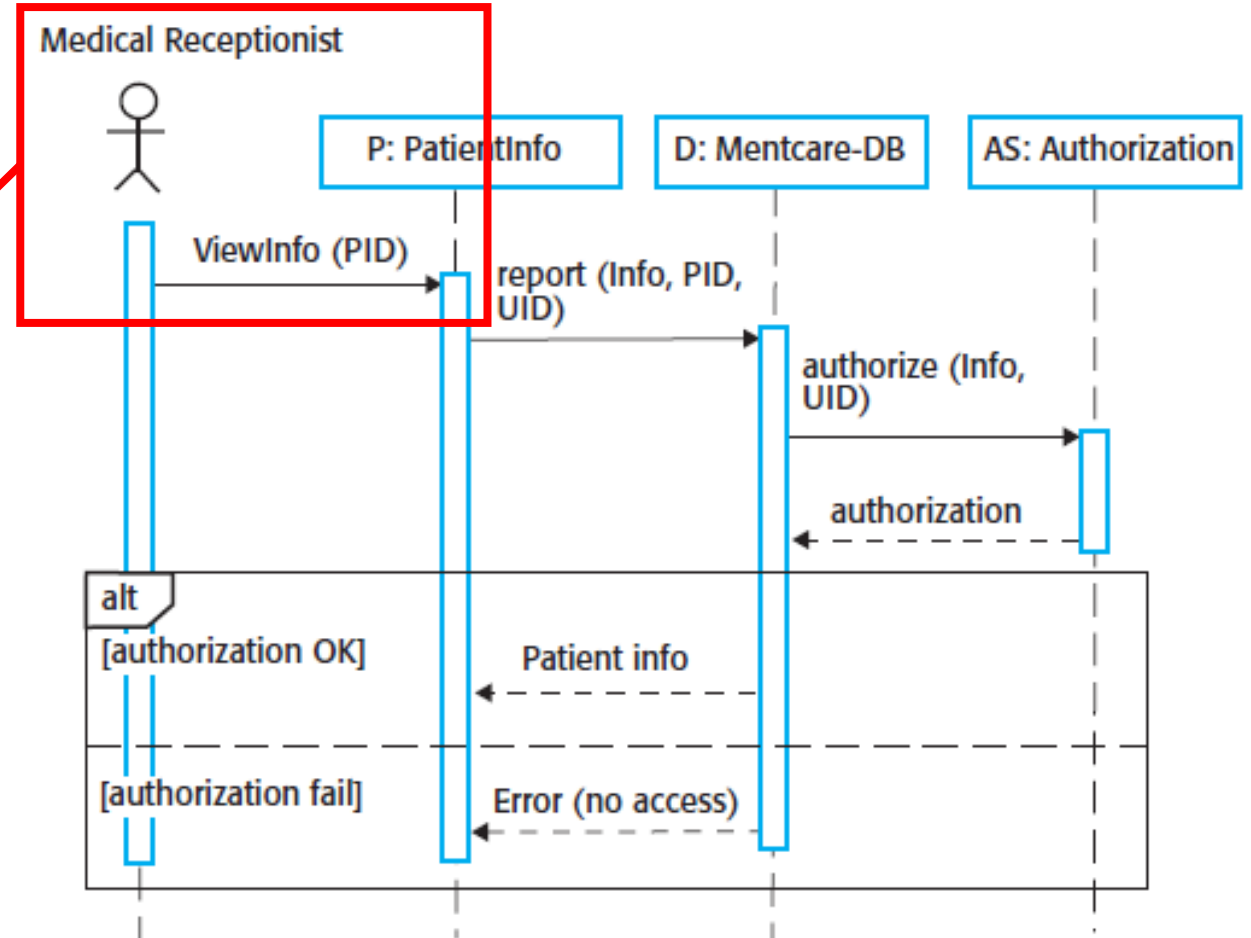


Fig: Sequence Diagram for View Patient Information

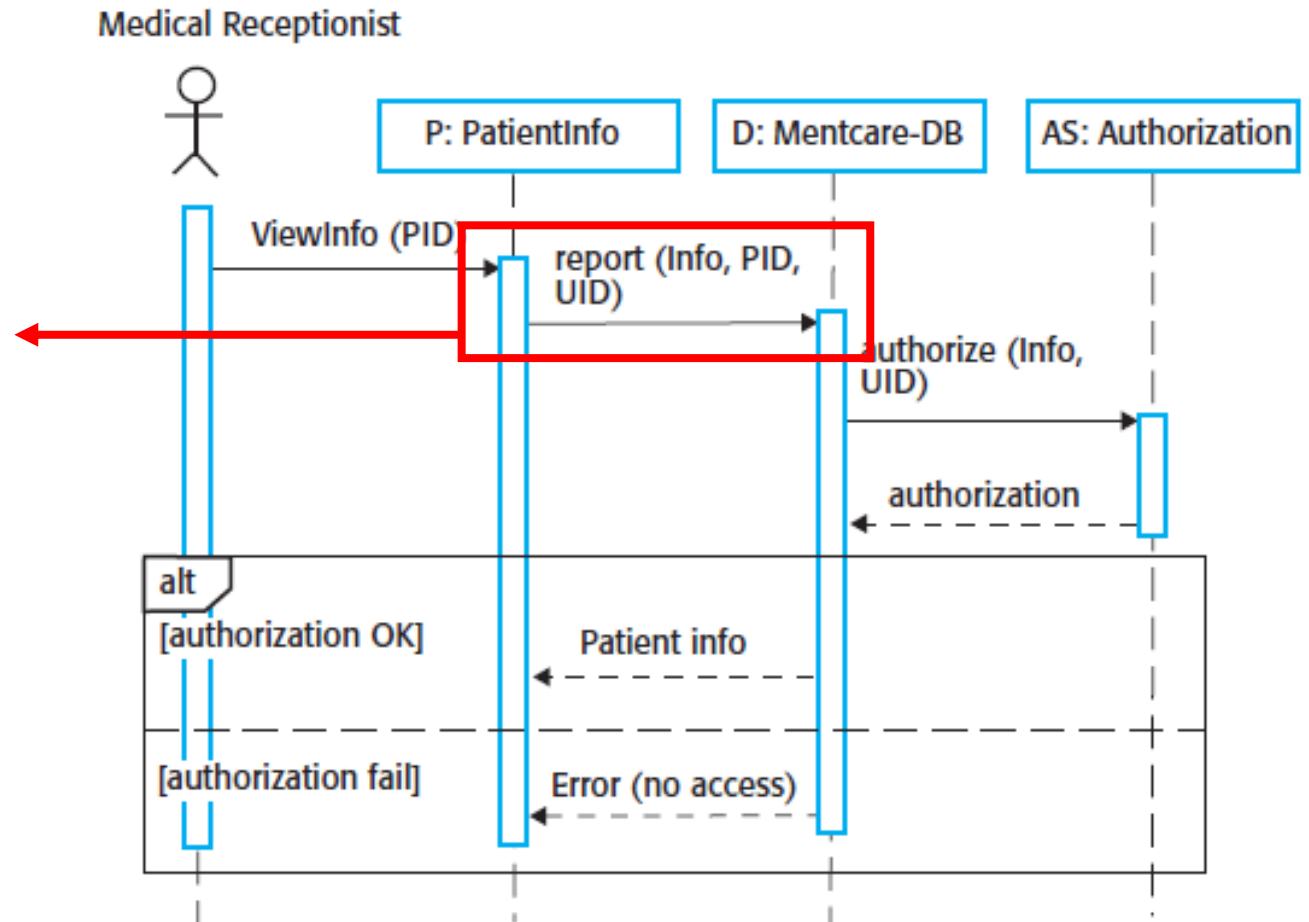
# Sequence Diagram (Read as)

1. The medical receptionist **triggers the ViewInfo method** in an instance **P of the PatientInfo object** class, **supplying the patient's identifier, PID** to identify the required information. P is a user interface object, which is displayed as a form showing patient information.



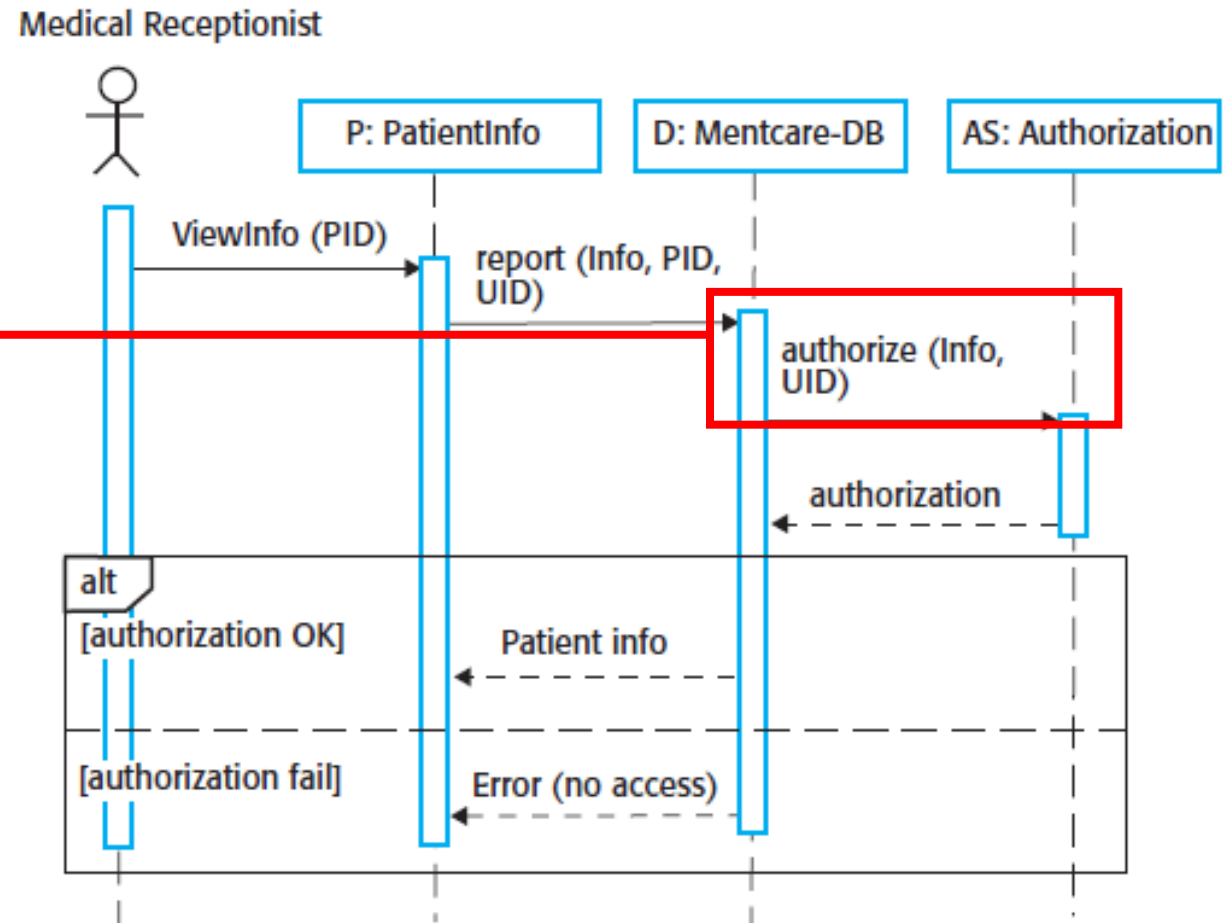
# Sequence Diagram (Read as)

2. The instance P calls the database to return the information required, supplying the receptionist's identifier, UID to allow security checking. (At this stage, it is not important where the receptionist's UID comes from.)



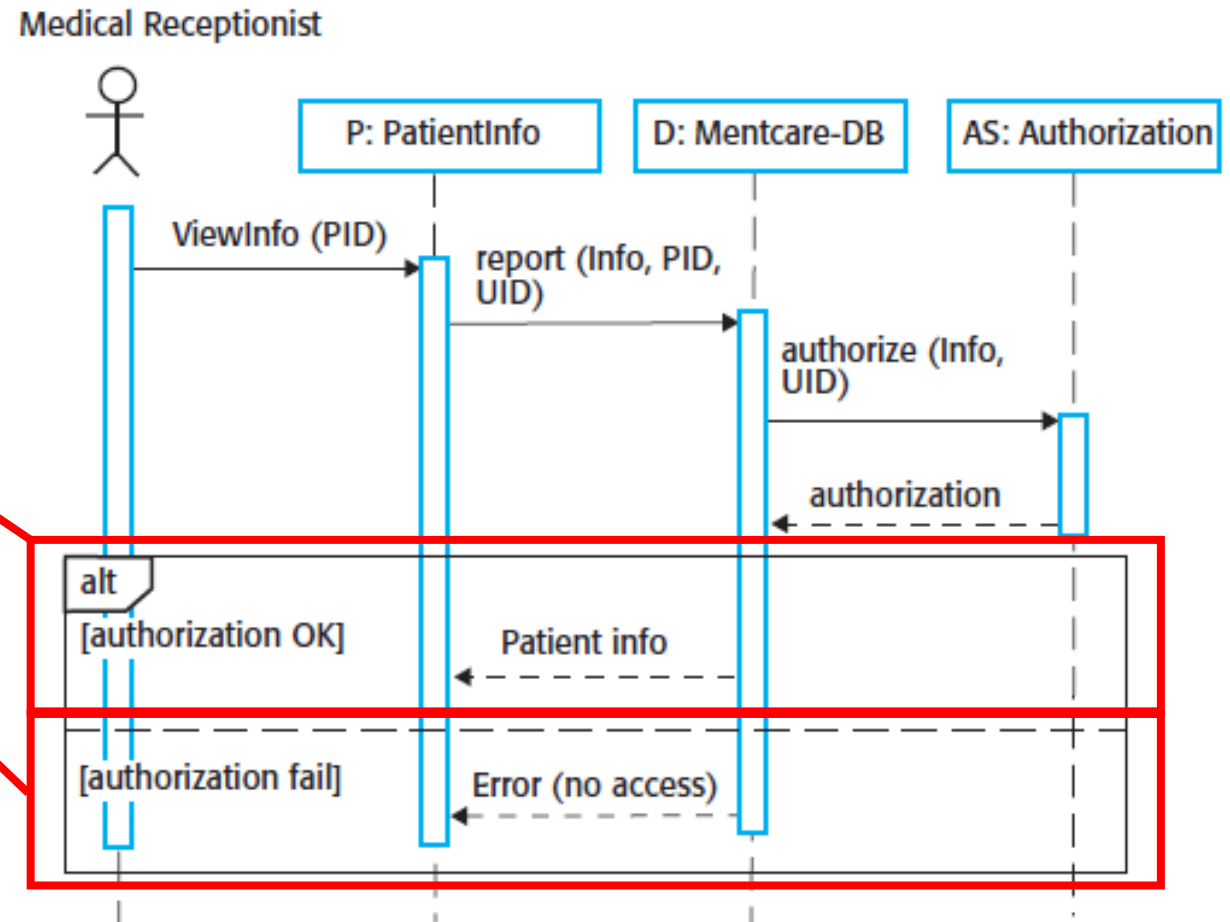
# Sequence Diagram (Read as)

3. The database checks with an authorization system that the receptionist is authorized for this action.



# Sequence Diagram (Read as)

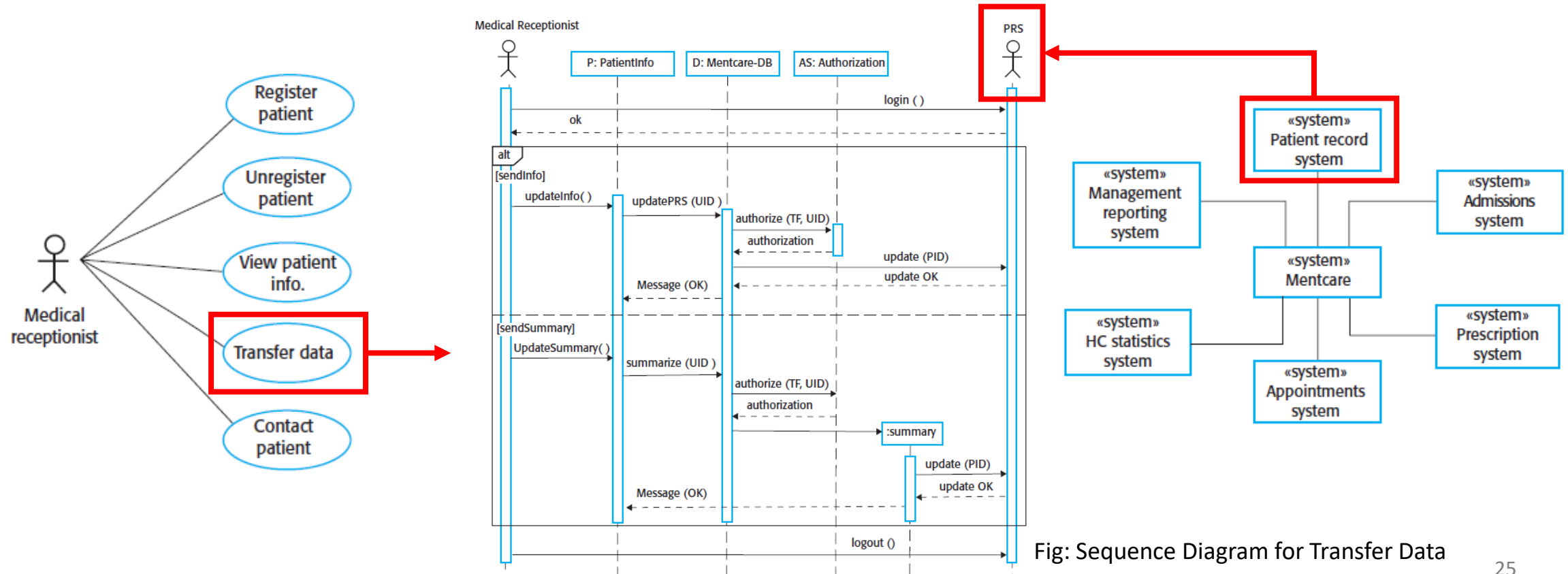
4. If authorized, the patient information is returned and is displayed on a form on the user's screen. If authorization fails, then an error message is returned. The box denoted by "alt" in the top-left corner is a choice box indicating that one of the contained interactions will be executed. The condition that selects the choice is shown in square brackets.





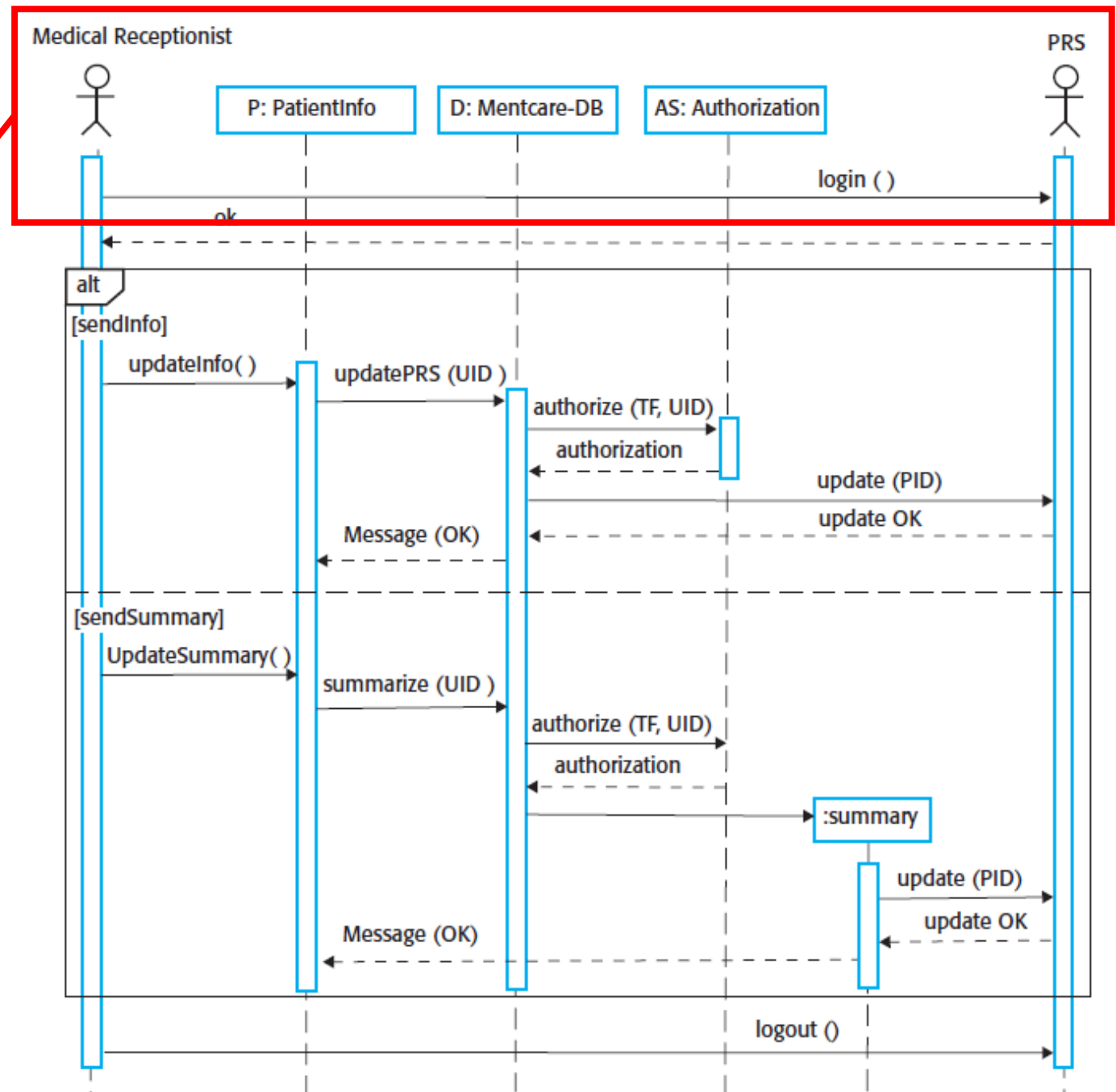
## Sequence Diagram + Context Model (More Complicated with Two Additional Features)

- **Direct communication between the actors in the system** and the creation of objects as part of a sequence of operations.
- The object of type Summary is created to hold the summary data that is to be uploaded to a national **patient records system** (PRS).



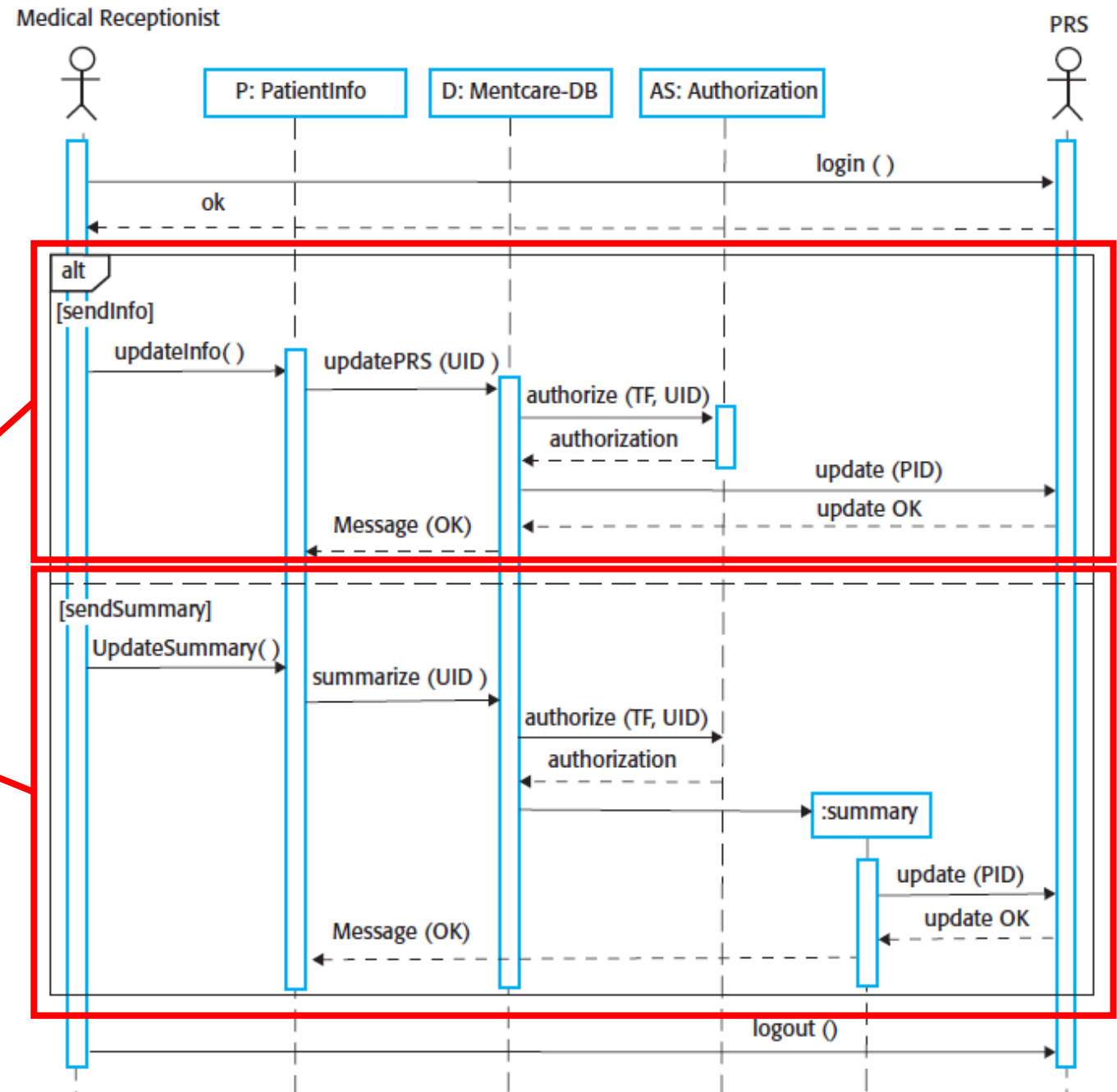
## Sequence Diagram for “Transfer Data” Use Case

1. The receptionist **logs on** to the PRS.



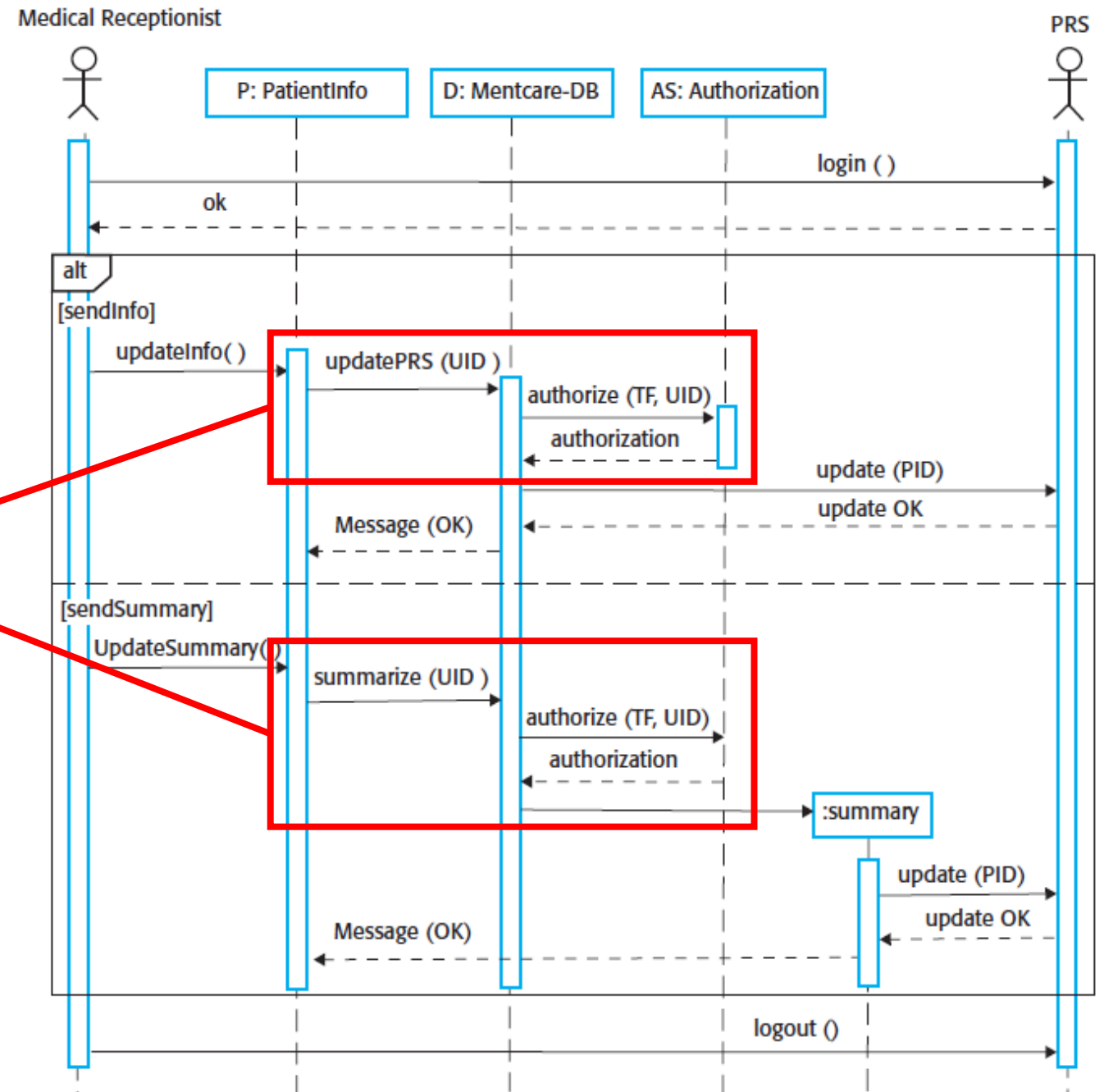
## Sequence Diagram for “Transfer Data” Use Case

2. **Two options** are available (as shown in the “alt” box). These allow the **direct transfer of updated patient information** from the Mentcare database to the PRS and the **transfer of summary health data** from the Mentcare database to the PRS.



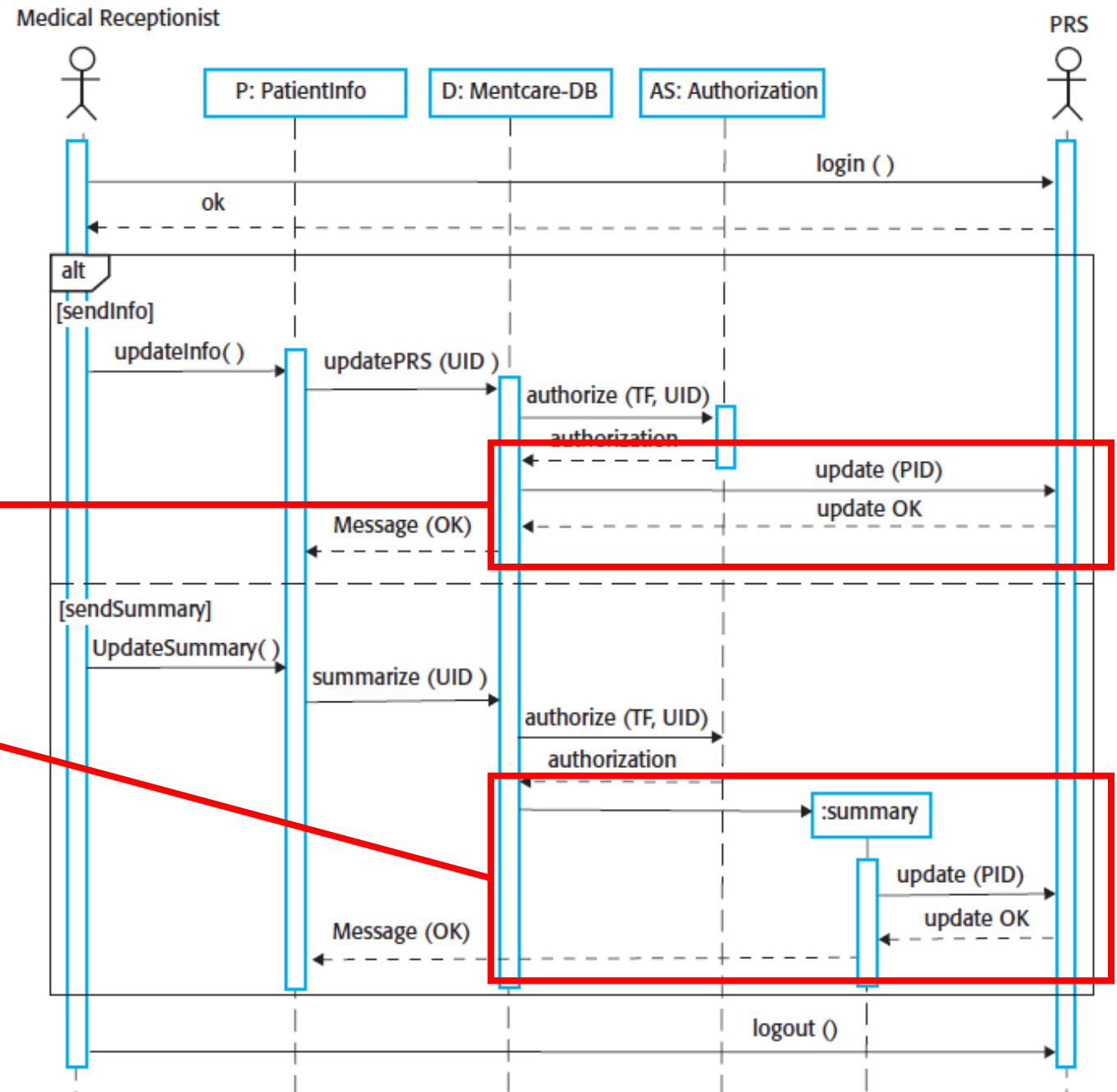
## Sequence Diagram for “Transfer Data” Use Case

3. In each case, the receptionist's permissions are checked using the authorization system.



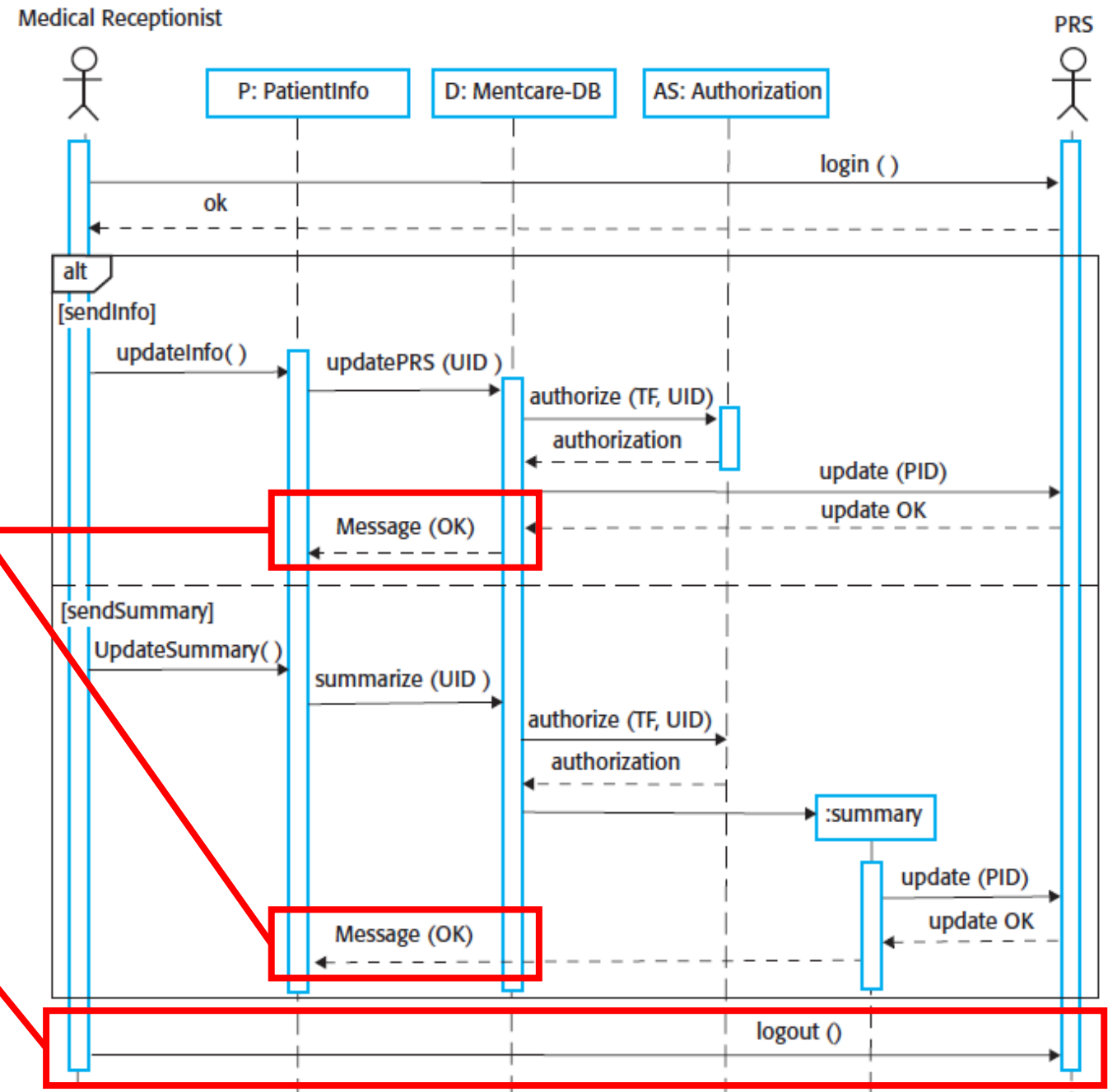
## Sequence Diagram for “Transfer Data” Use Case

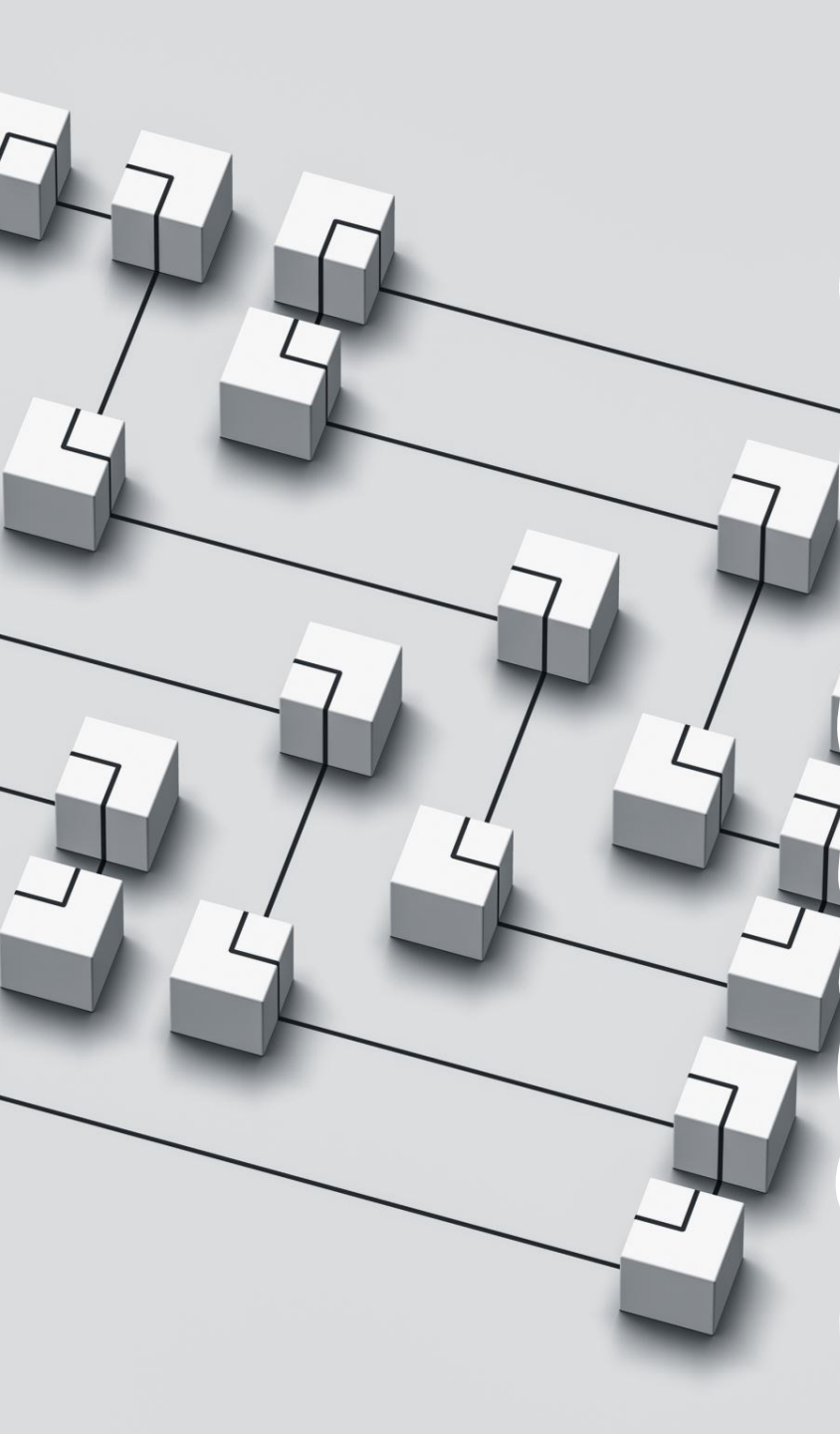
4. Personal information may be transferred directly from the user interface object to the PRS. Alternatively, a summary record may be created from the database, and that record is then transferred.



## Sequence Diagram for “Transfer Data” Use Case

- On completion of the transfer, **the PRS issues a status message and the user logs off.**

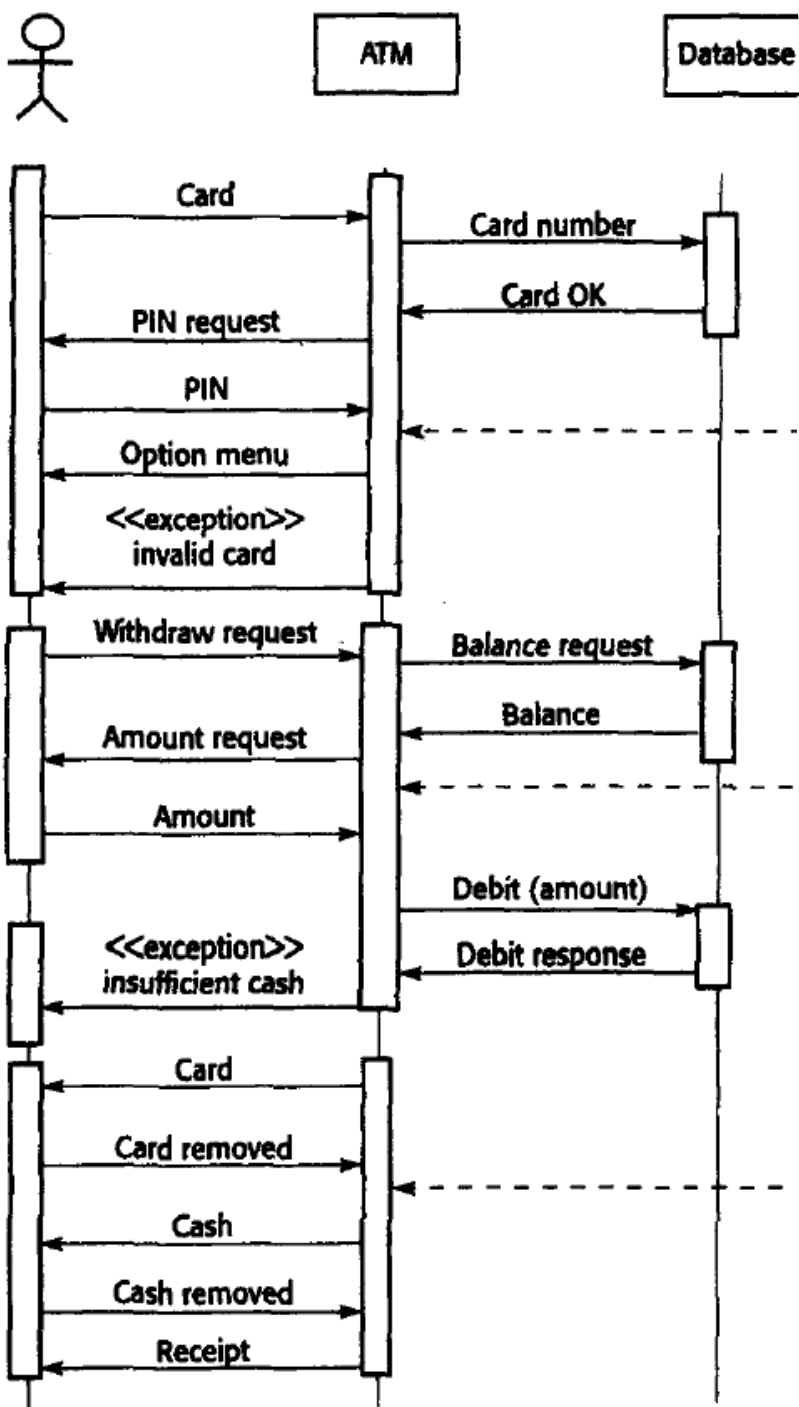




# Sequence Diagram in Different Stages

---

- If you're not using sequence diagrams for code generation or detailed documentation purposes, it's **unnecessary to include every interaction** in these diagrams.
- When you develop system models **early in the development process** to aid requirements engineering and high-level design, there will be **numerous interactions that rely on implementation decisions**.




# Sequence Diagram (Classic Example – ATM)

- Using an ATM

(Kevin Curran, David King, (2008) "Investigating the human computer interaction problems with automated teller machine navigation menus", Interactive Technology and Smart Education," vol. 5, Iss:1, pp. 59-79, DOI: [10.1108/17415650810871583](https://doi.org/10.1108/17415650810871583))





# Other UML Diagrams (mostly\* for later stages, like System Requirements (or System Specifications))

\*but can help in User Requirements

# UML – Key Diagram Types

Context Models  
Activity Diagrams



**Initial Requirements**

Could be Req., Spec. or Doc.

Use Case Diagrams  
\*Sequence Diagrams



**Refined Requirements**

Could be Req., Spec. or Doc.

Class Diagrams  
\*Sequence Diagrams  
State Diagrams



**Architecture Design**

Of Code – tho more for Spec. and Doc.

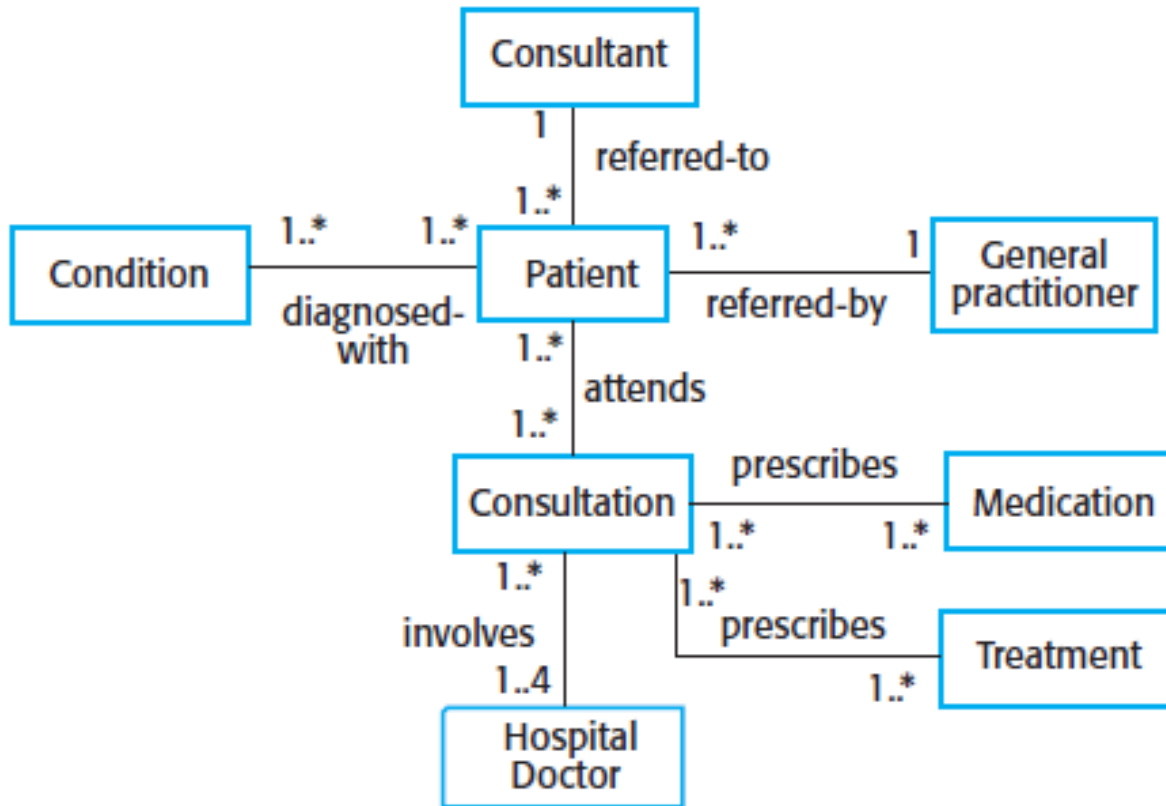


Fig: Classes and Associations in the Mentcare System

## Class Diagrams

- Specify the classes in **object-oriented** system model.
- To show classes in a system and the **association** between these classes.
- **Objects represent something in the real world**, such as patient, a prescription or a doctor.
- Class diagrams in UML can be expressed at varying levels of detail.
- During the initial stages of model development
  - To examine the environment, recognize the fundamental objects, and represent them as classes.

# Class Diagrams

- A class can be extended to show attributes (fields) and operations (methods)
  - The **name of the object class** is in the top section.
  - The **class attributes (fields)** are in the middle section. This include the attribute names, and optionally, their types (not shown here).
  - The **methods** associated with object class are in the lower section of the rectangle, optionally, their types (not shown here).

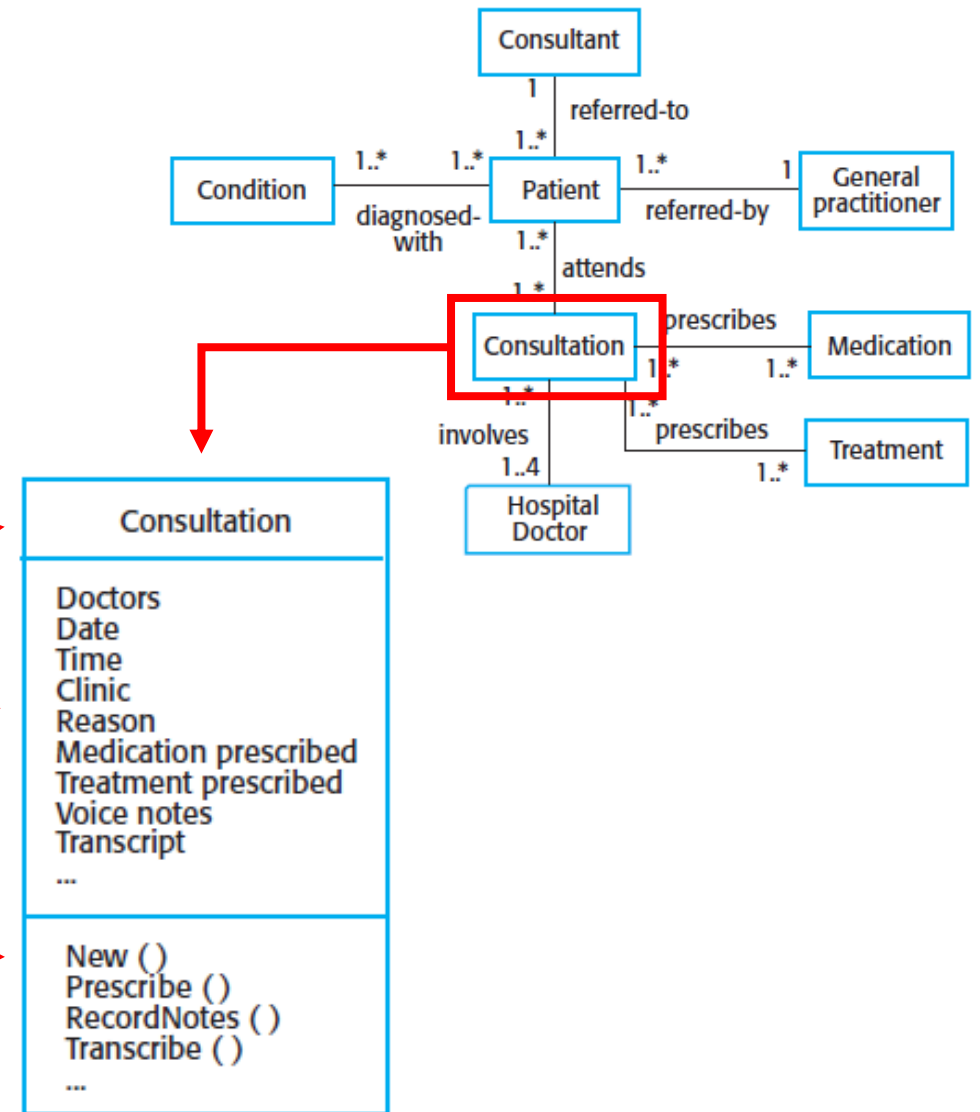
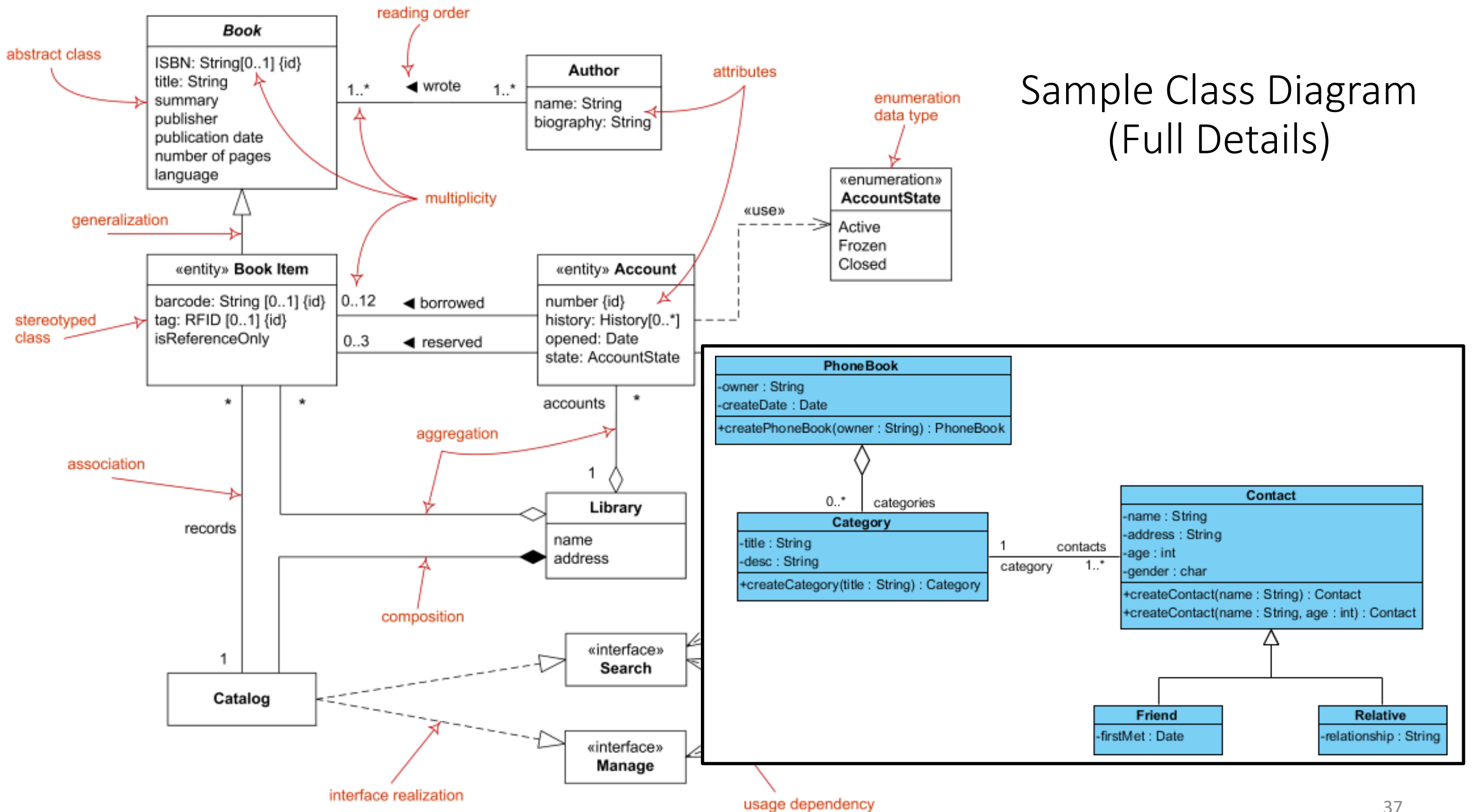


Fig: A Consultation Class

# Sample Class Diagram (Full Details)



# Class Diagrams (Generalization)

- Class diagram has a **specific type of association to denote generalization** with an arrowhead pointing up to the more general class.
- General practitioners and hospital doctors can be **generalized as doctors**.
  - Hospital doctor and general practitioner are types of doctors.
- **Three types of Hospital Doctor:**
  - Those who have just graduated from medical school and have to be supervised (**Trainee Doctor**);
  - Those who can work unsupervised as part of a consultant's team (**Registered Doctor**);
  - **Consultants**, who are senior doctors with full decision-making responsibilities.
- Not associated through generalization:
  - E.g., trainee doctor is not a type of general practitioner.

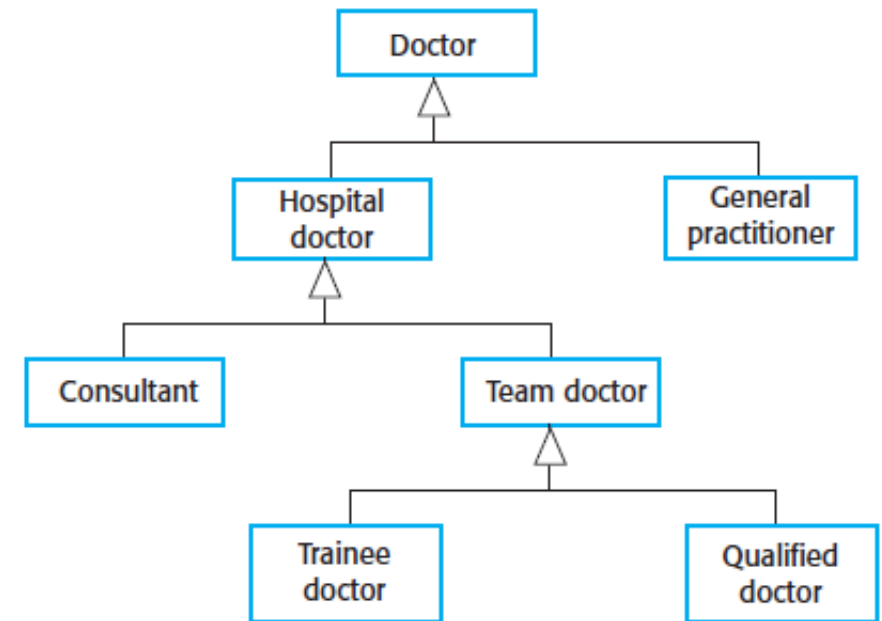


Fig: A Generalization Hierarchy

# Class Diagrams

## (Generalization with Added Detail)

- In a generalization, the **attributes and operations linked with higher-level classes are inherited by the lower-level classes.**
- These lower-level classes, known as **subclasses**, **inherit attributes and operations from their superclasses** and may **introduce additional specific attributes and operations.**
- For instance,
  - All doctors (including hospital doctor and general practitioner) have a name and phone number, and they can perform register and de-register actions.
  - All hospital doctors have a staff number and carry a pager (not including other doctors and general practitioner).
  - General practitioners have an individual practice name and address (not including other doctors and hospital doctors).

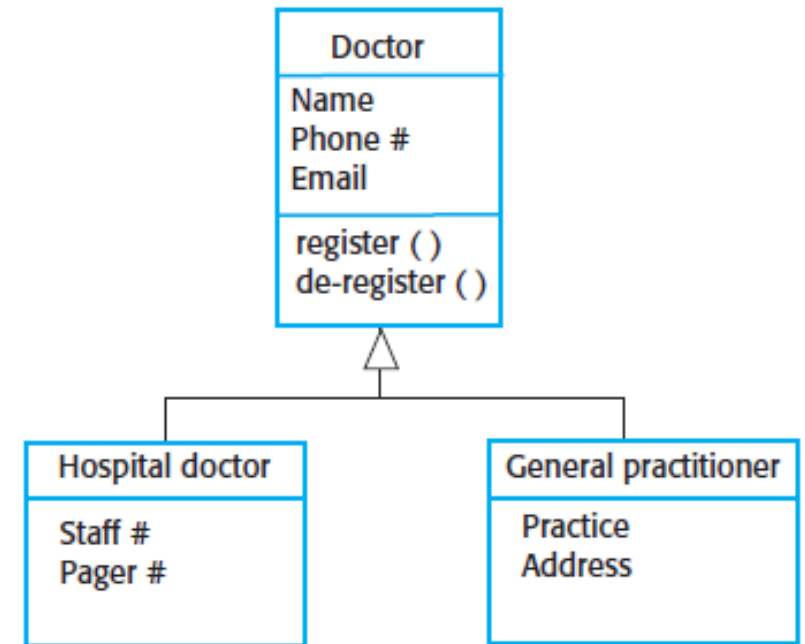


Fig: A Generalization Hierarchy with Added Detail

# UML – Key Diagram Types

Context Models  
Activity Diagrams



**Initial Requirements**

Could be Req., Spec. or Doc.

Use Case Diagrams  
\*Sequence Diagrams



**Refined Requirements**

Could be Req., Spec. or Doc.

Class Diagrams  
\*Sequence Diagrams  
State Diagrams



**Architecture Design**

Of Code – tho more for Spec. and Doc.



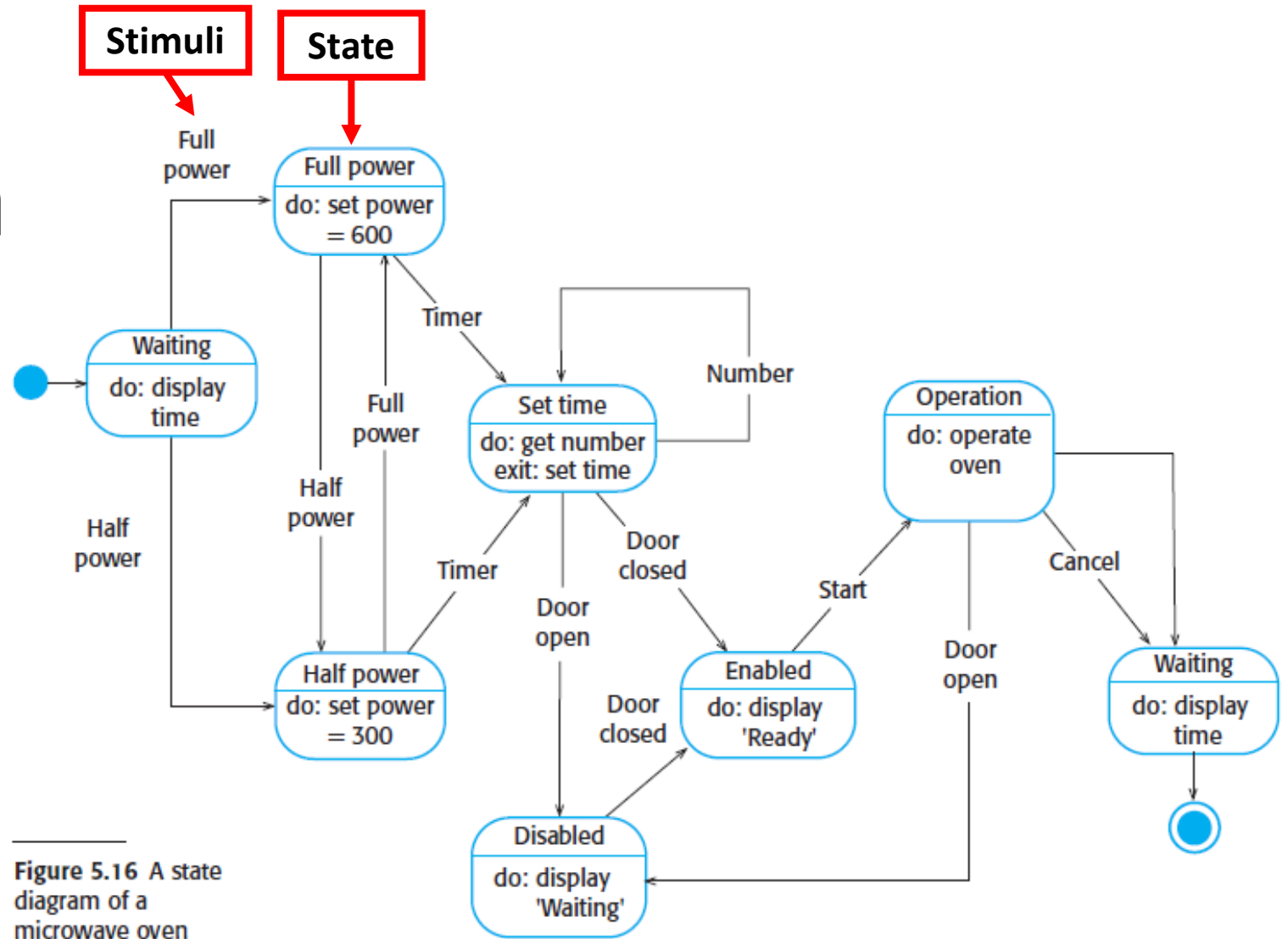
# State Diagrams

---

- Illustrates the system's **reactions to both external and internal events**, operating under the **assumption that a system has a finite set of states**, and **events (stimuli) can induce transitions between these states**.
- For instance, a basic microwave oven might feature a switch to toggle between full or half power, a numeric keypad to input cooking time, a start/stop button, and an alphanumeric display.
  - For safety reasons, the oven should be inactive when the door is open.
  - It may also incorporate a straightforward display to convey alerts and warning messages.

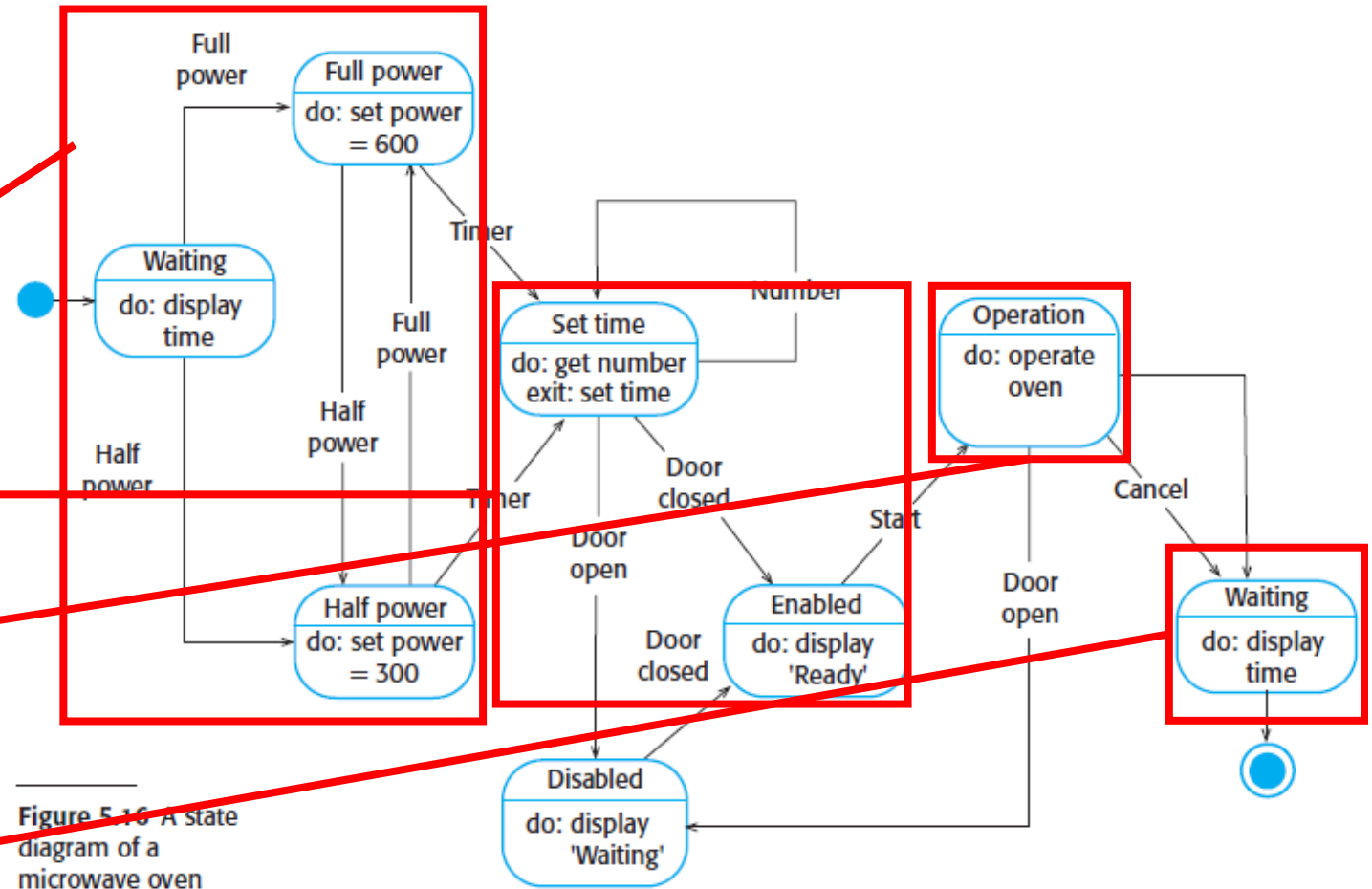
# State Diagram

- In a state diagram:
  - **Rounded rectangles represent system states** – may include a brief description (following “do”) of the actions taken in that state.
  - **Labeled arrows represent stimuli** that force a transition from one state to another.
  - Indicate **start and end states** using filled circles.



# State Diagrams

- The system starts in a waiting state and responds initially to either the full-power or the half-power button (user can choose by pressing the button).
- The time is set and, if the door is closed, the Start button is enabled.
- Pushing this button starts the oven operation, and cooking takes place for the specified time.
- This is the end of the cooking cycle, and the system returns to the waiting state.



# State Diagrams (A State)

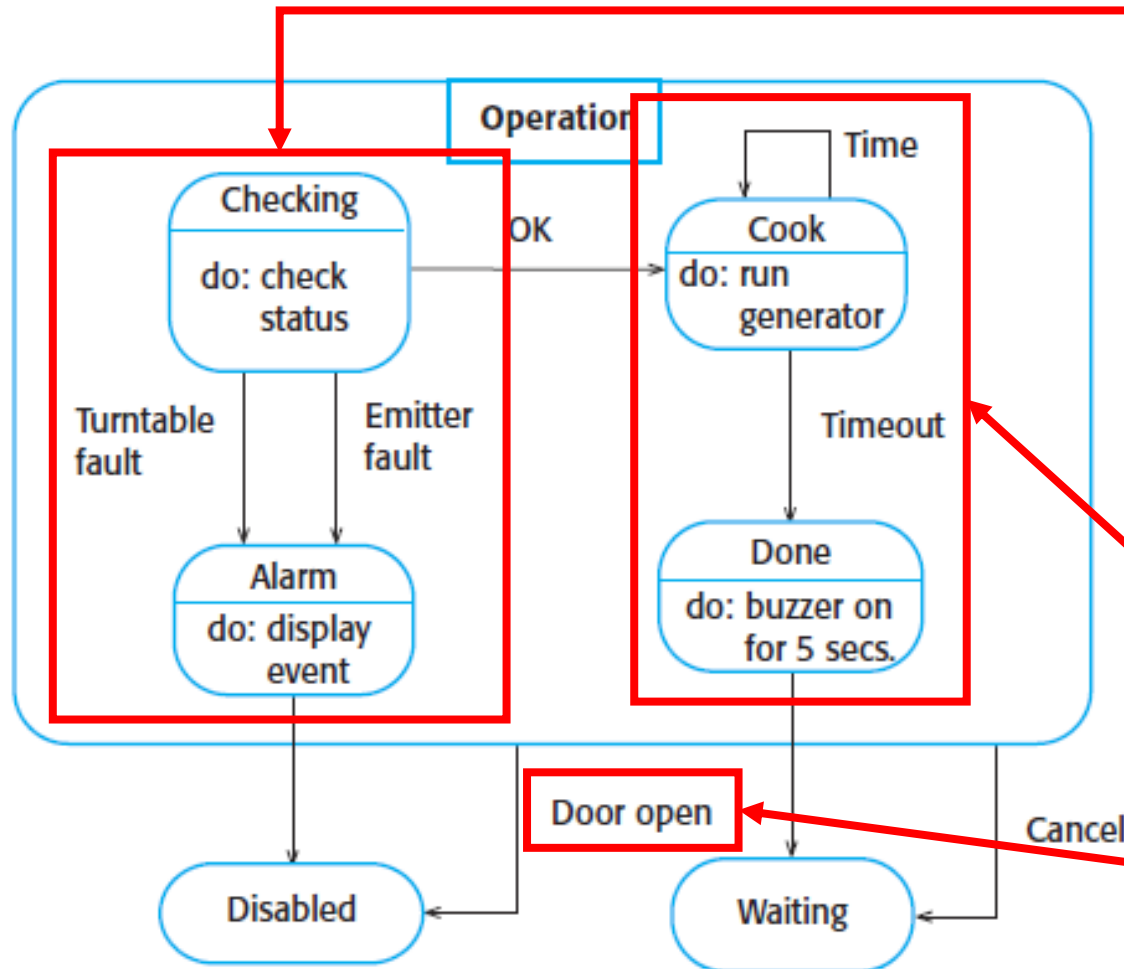


Fig: A State Model of the Operation State

- The **Operation** state includes several substates.
  - Operation starts with a status check and that if any problems are discovered an alarm is indicated and operation is disabled.
  - Cooking involves running the microwave generator for the specified time; on completion, a buzzer is sounded.
  - If the door is opened during operation, the system moves to the disabled state.

| State       | Description  |
|-------------|--|
| Waiting     | The oven is waiting for input. The display shows the current time.   |
| Half power  | The oven power is set to 300 watts. The display shows "Half power."  |
| Full power  | The oven power is set to 600 watts. The display shows "Full power."  |
| Set time    | The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.  |
| Disabled    | Oven operation is disabled for safety. Interior oven light is on. Display shows "Not ready."   |
| Enabled     | Oven operation is enabled. Interior oven light is off. Display shows "Ready to cook."  |
| Operation   | Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows "Cooking complete" while buzzer is sounding. |
| Stimulus    | Description  |
| Half power  | The user has pressed the half-power button.  |
| Full power  | The user has pressed the full-power button.  |
| Timer       | The user has pressed one of the timer buttons.   |
| Number      | The user has pressed a numeric key.  |
| Door open   | The oven door switch is not closed.  |
| Door closed | The oven door switch is closed.  |
| Start       | The user has pressed the Start button.   |
| Cancel      | The user has pressed the Cancel button.  |

# State Diagrams

- Can use a table to list the states and events that show state transitions, alongside a description for each state and event.

Fig: States and Stimuli for the Microwave Oven

# How to Choose Different Diagrams?

---

1. In the initial steps of requirements:
  - What is or is not part of the system? -- **Context Models**: *Show how a system that is being modeled is positioned in an environment with other systems and processes. Help to define the **boundaries** of the system to be developed.*
  - How are the systems used? -- **Activity Diagrams**: *Used to model the processing of data, where each activity represents one process step.*
2. Next, how to model the **interactions** between users, internal and external systems?
  - **Use Case Diagrams** are drafts: *Describe interactions between a system and external actors.*
  - **Sequence Diagrams** are used to explain the specific use case: *Add more information to these interactions by showing interactions between system objects.*
3. Most importantly, how to design the **system architecture**?
  - Static view of the software system can be modeled as **Class diagrams**: *Define the static structure of classes in a system and their associations.*
  - Dynamic view can be modeled as **Sequence Diagrams** if the system are driven by data.
  - If the systems are driven by events, it can be modeled as **State Diagrams**: *Used to model a system's behavior in respond to internal or external events.*

# Summary

---

1. What are the **reasons** for utilizing Requirements Modeling?
  - It facilitates a complex decision-making process.
  - It encompasses five common diagrams out of the 13 UML diagrams.
2. **Avoids the practice of attempting to show everything in a single diagram.**
  - It focuses on highlighting essential elements while disregarding some details.
  - In general, a system typically includes:
    - One use case diagram (UCD).
    - Several sequence diagrams, with each corresponding to a use case (in an ideal scenario) from the UCD.
    - Likewise, numerous activity diagrams, each corresponding to a use case (in an ideal scenario) from the UCD.
    - One class diagram.
    - One state diagram.

# Optional Readings

---

- UML Diagrams



AND  
YOU