COMP 433 Software Engineering

Module 4: *System Modeling*

Ahmed Tamrawi

atamrawi atamrawi.github.io ahmedtamrawi@gmail.com

Acknowledgment Notice

Part of the slides are based on content from Ian Somerville's Software Engineering book.

System Modeling

- **System modeling** is the *process of developing abstract models of a system, with each model presenting a different view or perspective of that system*.
- System modeling has now come to mean representing a system using graphical notation, which is now almost always based on notations in the *Unified Modeling Language (UML).*
- System modelling **helps the analyst to understand the functionality** of the system and **models are used to communicate with customers**.

Existing and Planned System Models

- Models of the existing system are used during requirements engineering.
  - They *help clarify what the existing system does* and can be used as a *basis for discussing its strengths and weaknesses*.
  - These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to *help explain the proposed requirements to other system stakeholders*. Engineers use these models to *discuss design proposals and to document the system for implementation*.

Use of Graphical Models

- As a means of facilitating discussion about an existing or proposed system
    - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an existing system
    - Models should be an accurate representation of the system but need not be complete.
- As a detailed system description that can be used to generate a system implementation
    - Models must be both correct and complete.

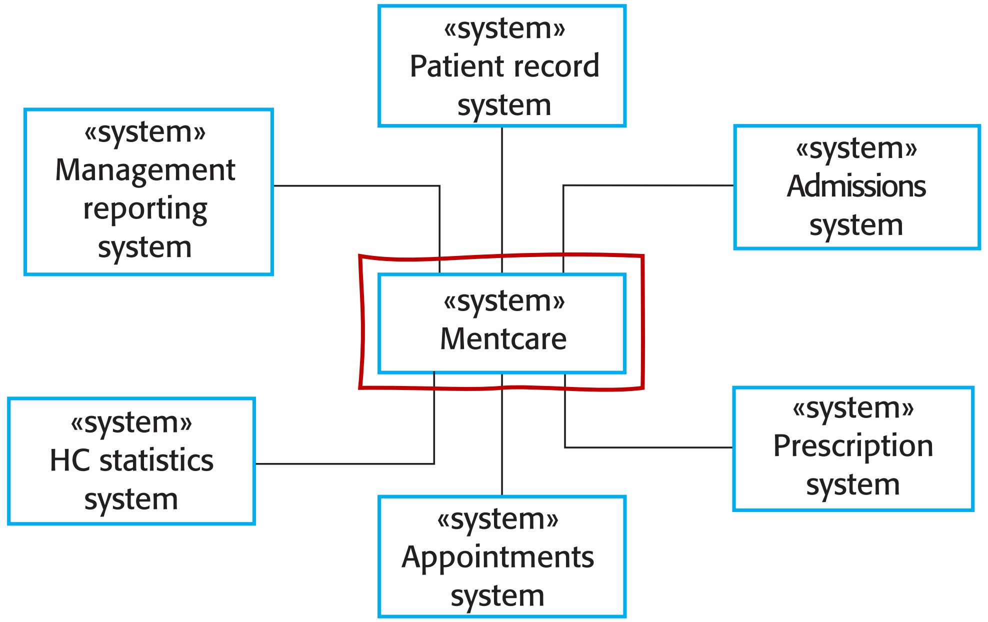
System Perspectives

- An **external "context" perspective**, where you model the context or environment of the system.
- An **interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.

Context Models

Context Models

- **Context models** are used to illustrate the operational context of a system - they show *what lies outside the system boundaries*.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- **Architectural models** show the system and its relationship with other systems.


«system» Patient record system
«system» Management reporting system
«system» Admissions system
«system» Mentcare
«system» HC statistics system
«system» Prescription system
«system» Appointments system

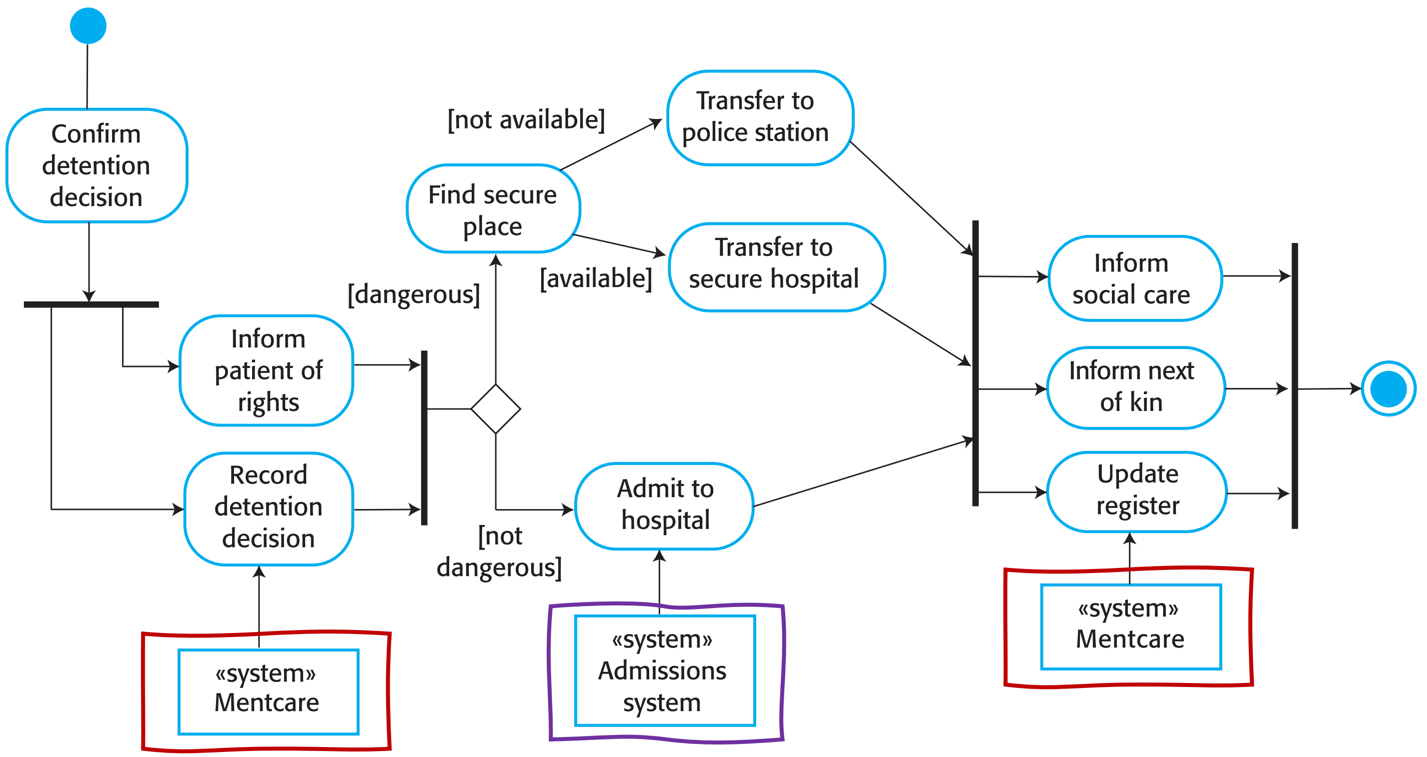
Context Models: *System Boundaries*

- System boundaries are established to *define what is inside and what is outside the system*. They show other systems that are used or **depend** on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase and/or decrease the influence or workload of different parts of an organization.

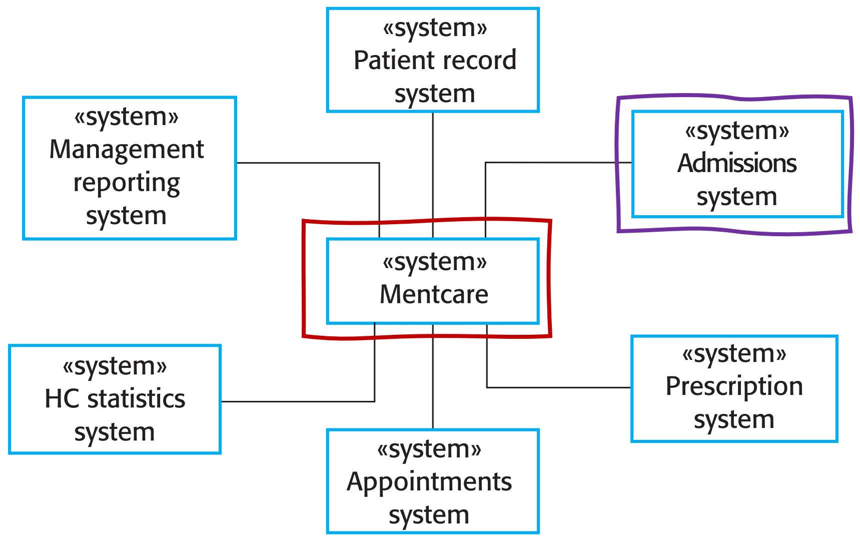
Context Models: *Process Perspective*

- Context models simply show *the other systems in the environment,* **not** how the system being developed is used in that environment.
- Process models reveal how the system being developed is used in broader business processes.
- **UML activity diagrams** may be used to define business process models.

Context Models: *Process Perspective*



Process Model of Involuntary Detention


«system» Patient record system
«system» Management reporting system
«system» Admissions system
«system» Mentcare
«system» HC statistics system
«system» Prescription system
«system» Appointments system

Context of the Mentcare System

Interaction Models

Interaction Models

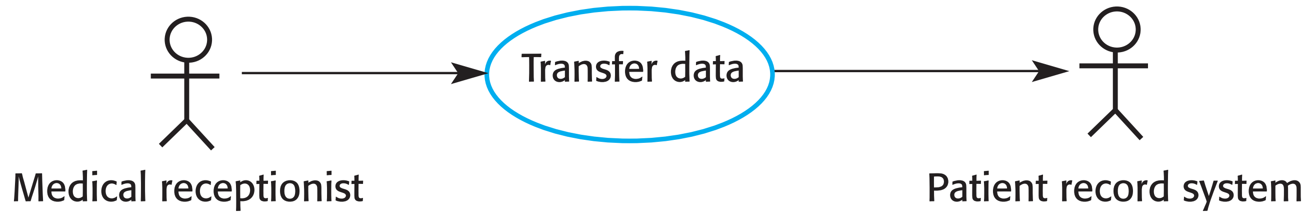
- Modeling user interaction is important as it helps to identify user requirements.
- Modeling **system-to-system interaction** highlights the communication problems that may arise.
- Modeling **component interaction** helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- **Use case diagrams** and **sequence diagrams** may be used for interaction modelling.

Interaction Models: *Use Case Modeling*

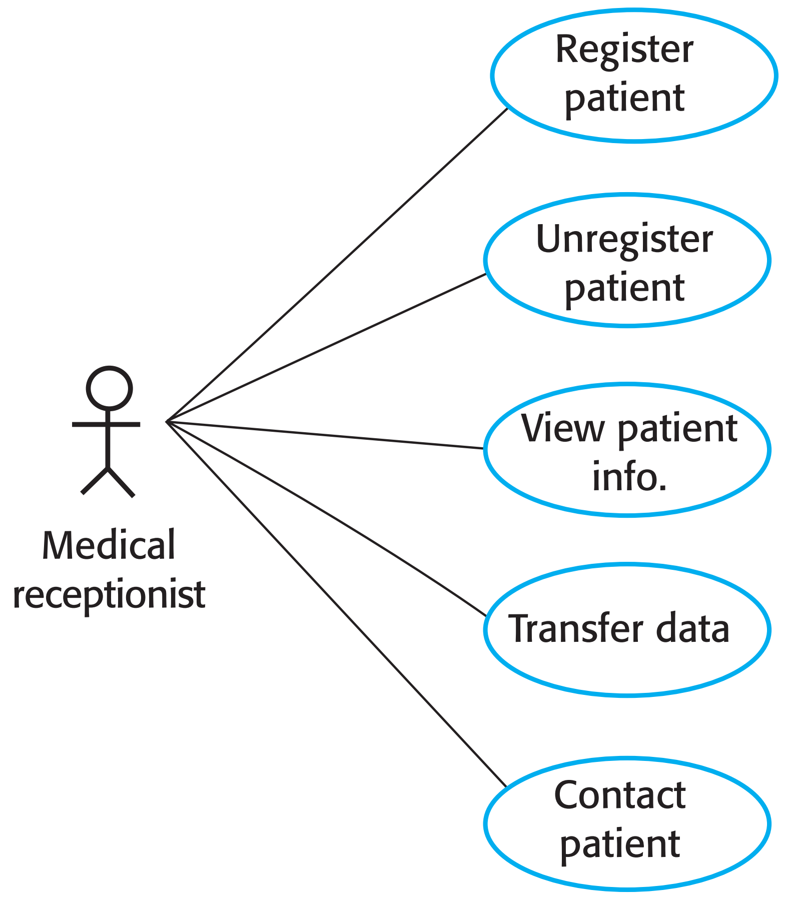
- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Actors in a use case may be people or other systems.
- Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.

Transfer-Data Use Case: *Mentcare System*

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Data	Patient's personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.



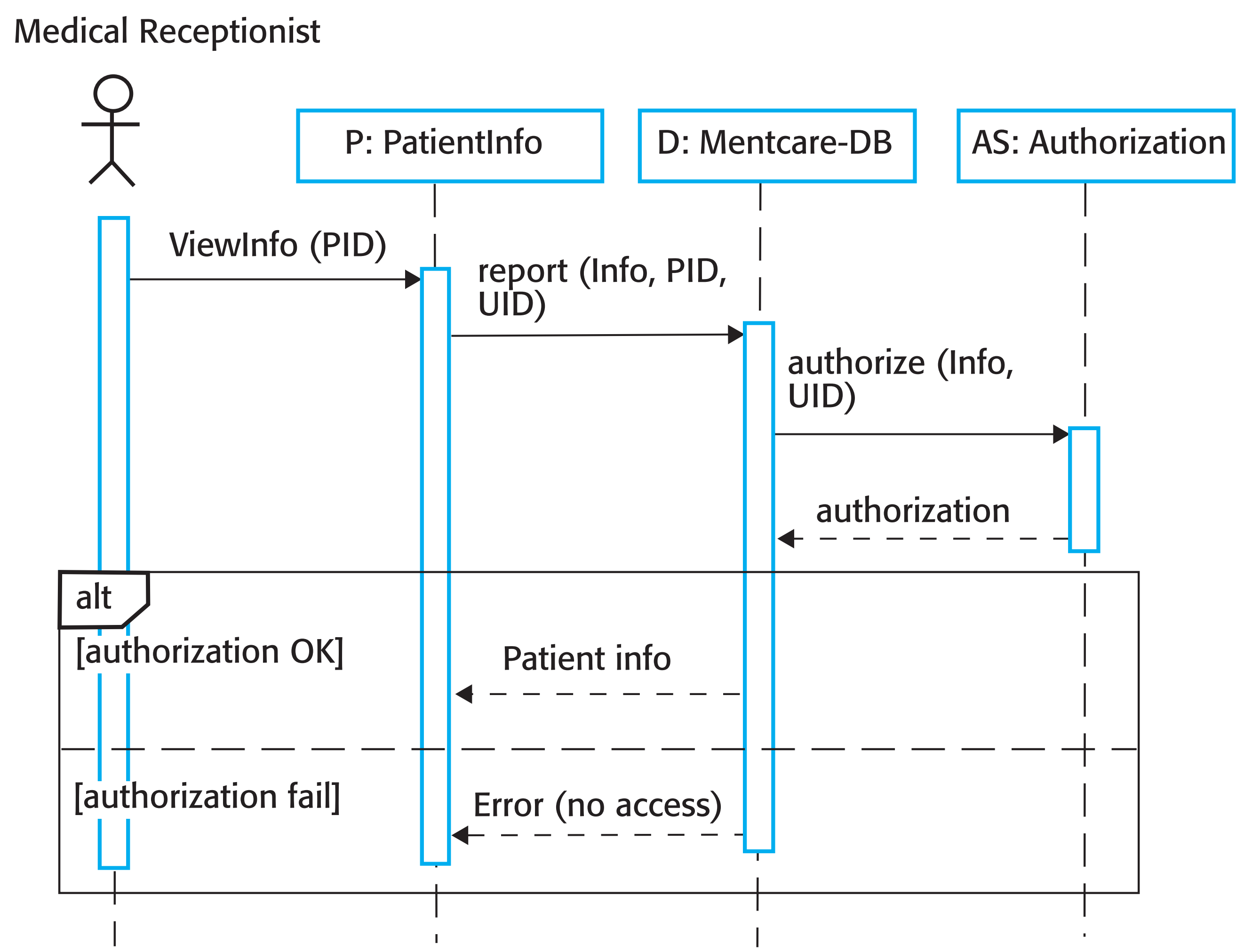
Use cases involving the role ‘*medical receptionist*’



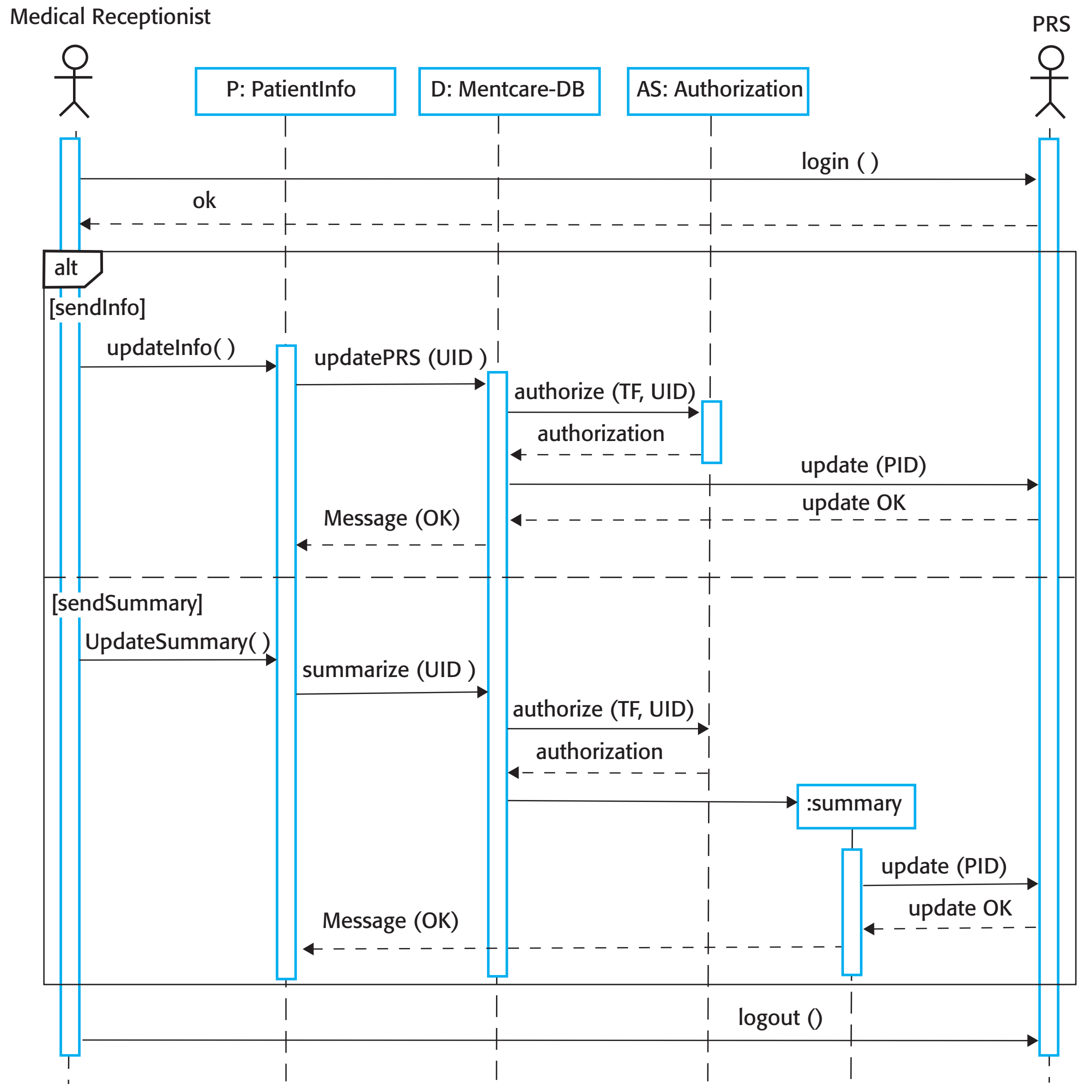
Interaction Models: *Sequence Diagrams*

- Sequence diagrams are part of the UML and are used to *model the interactions between the actors and the objects within a system.*
- A sequence diagram shows the **sequence of interactions** that take place during a particular use case or use case instance.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

Sequence Diagram for *View Patient Information*



Sequence Diagram for *Transfer Data*



Structural Models

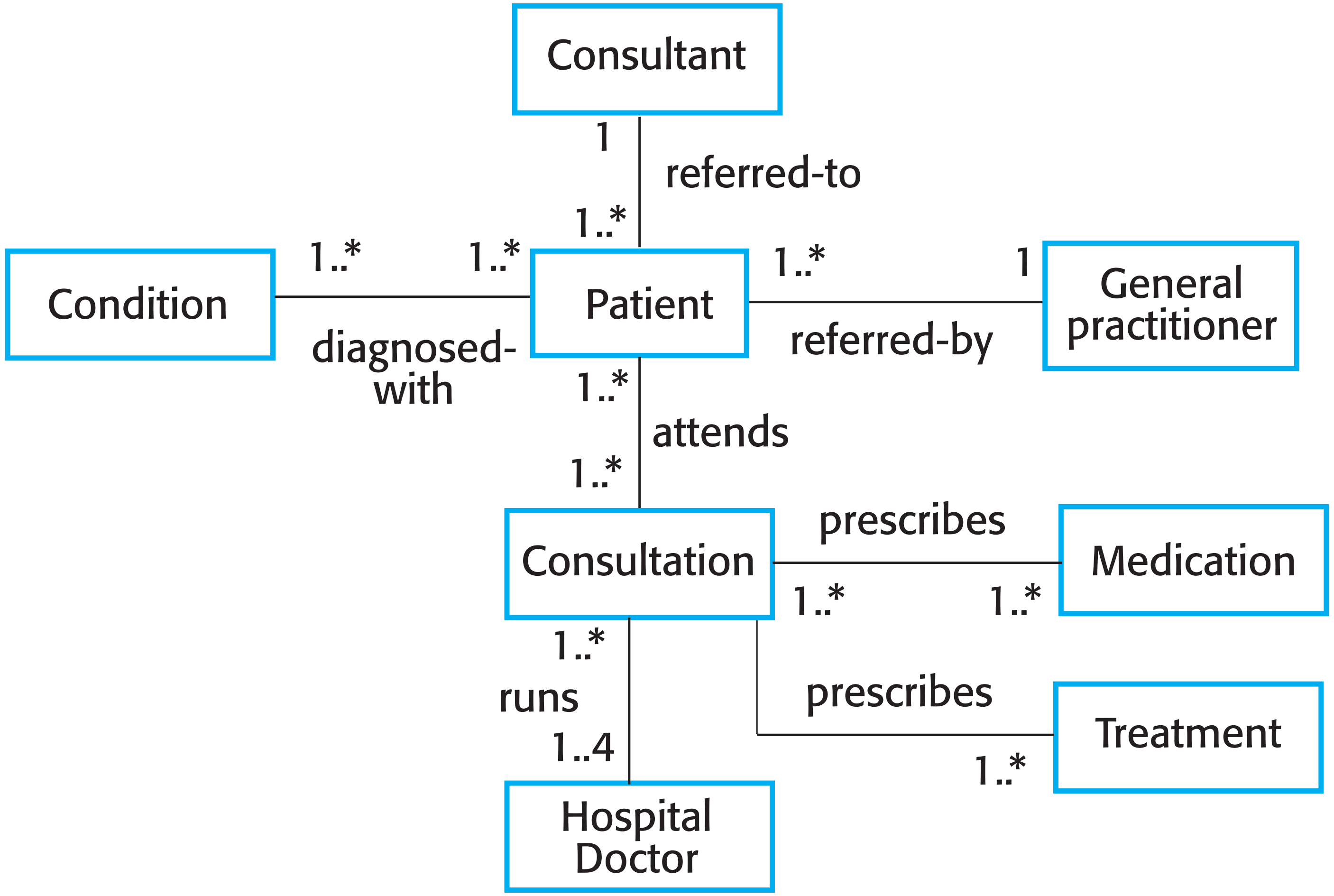
Structural Models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be **static models**, which show the *structure of the system design*, or **dynamic models**, which show the organization of the system when it is *executing*.
- You create structural models of a system when you are discussing and designing the **system architecture**. We will cover the topic of system architecture and architectural pattern in later lectures.

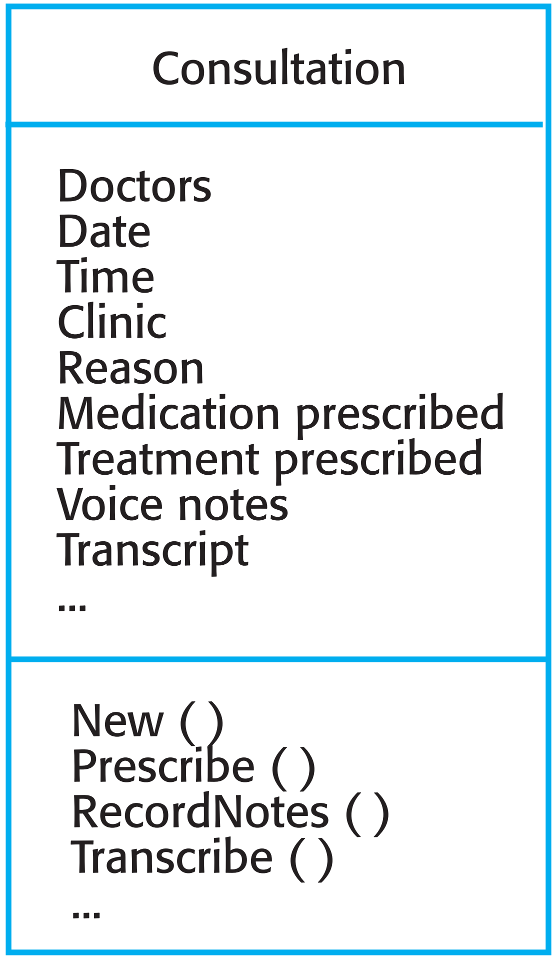
Structural Models: *Class Diagrams*

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An **association** is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

Classes and Associations in the MHC-PMS

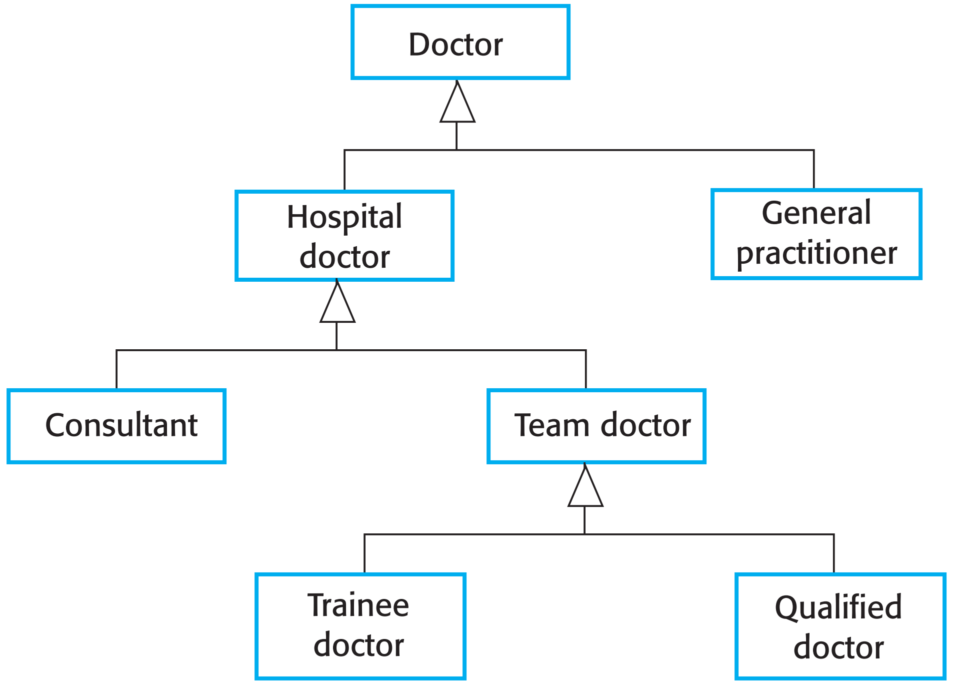


The Consultation Class



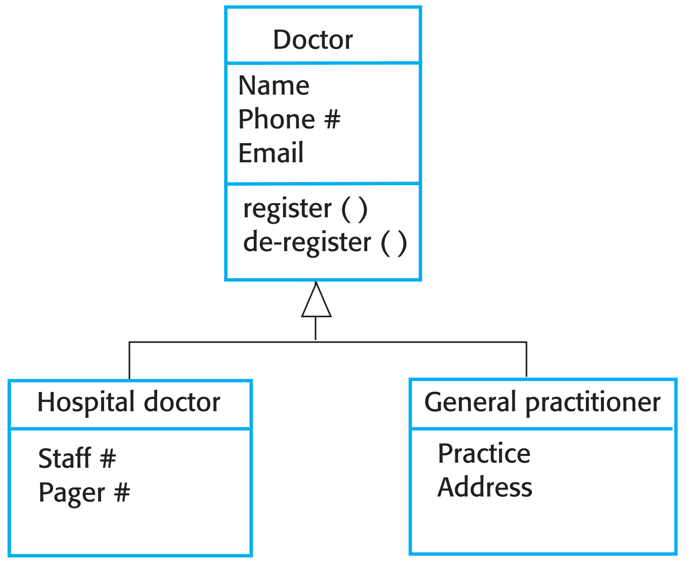
Generalization

- Generalization is an everyday technique that we use to manage complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- This allows us to infer that different members of these classes have some common characteristics. For example, squirrels and rats are rodents.



Generalization

- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization.
  - If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In object-oriented languages, such as Java, generalization is implemented using the **class inheritance** mechanisms built into the language.
- In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
  - The lower-level classes are subclasses inherit the attributes and operations from their super classes. These lower-level classes then add more specific attributes and operations.



Behavioral Models

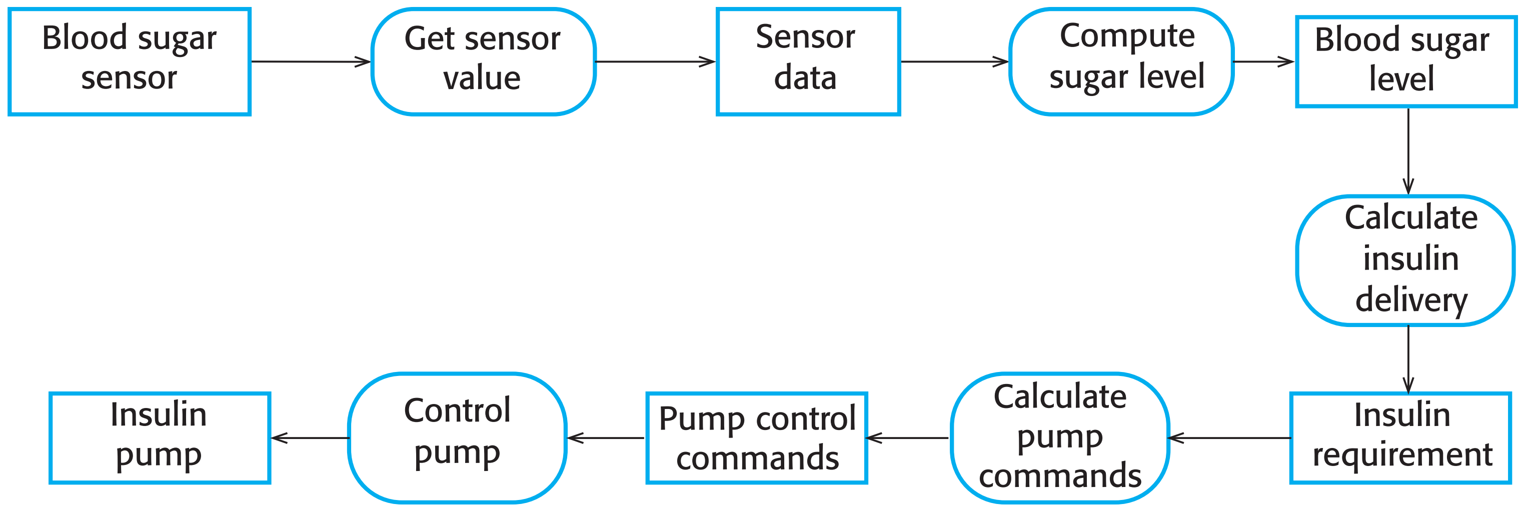
Behavioral Models

- Behavioral models are models of the **dynamic behavior** of a system as it is executing.
- They show what happens or what is supposed to happen when a system responds to a **stimulus** from its environment.
- You can think of these stimuli as being of two types:
  - Data - some data arrives that has to be processed by the system.
  - Events - some event happens that triggers system processing. Events may have associated data, although this is not always the case.

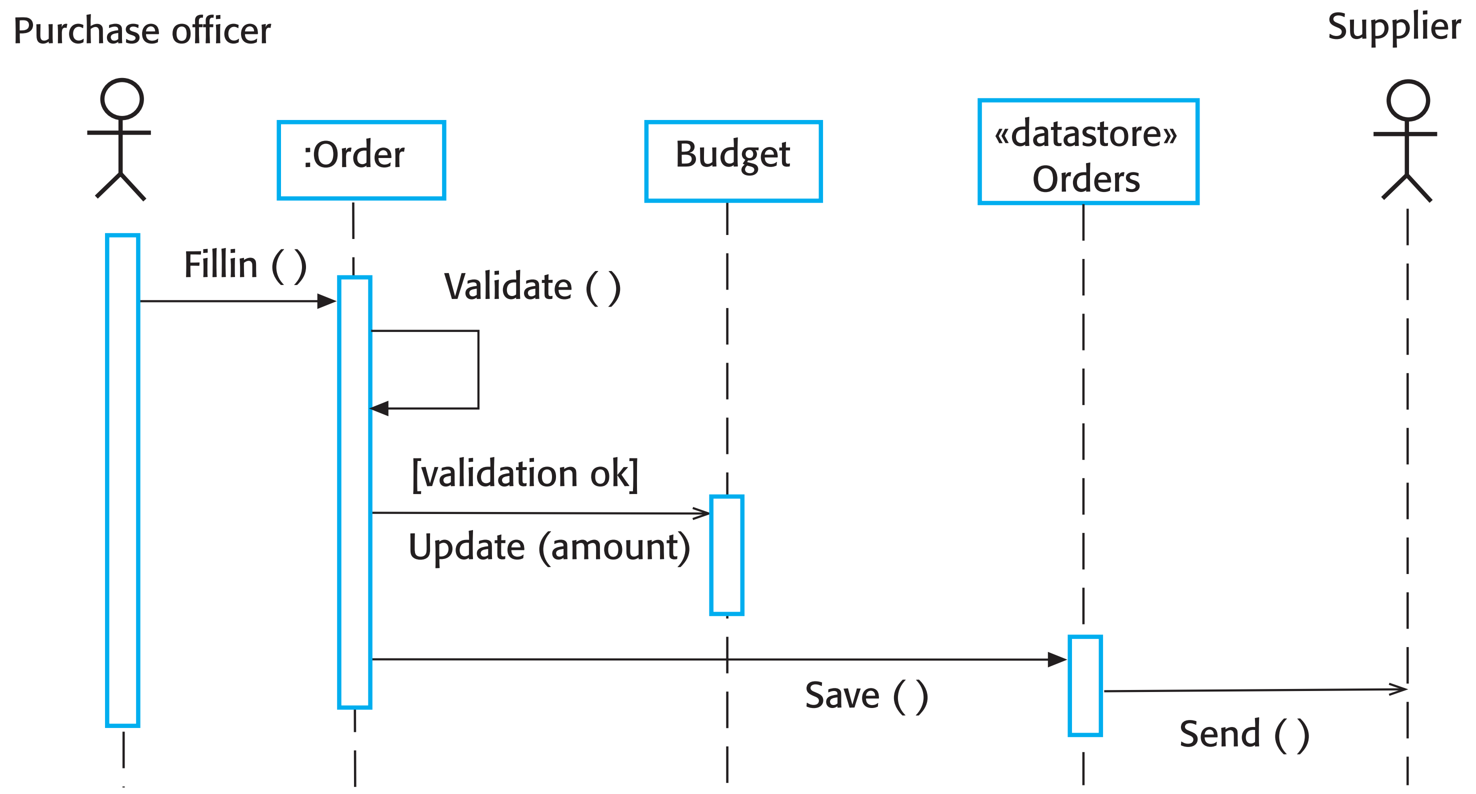
Behavioral Models: *Data-Driven Modeling*

- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated output.
- They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

Activity Model of the Insulin Pump's Operation



Order Processing



Behavioral Models: *Event-Driven Modeling*

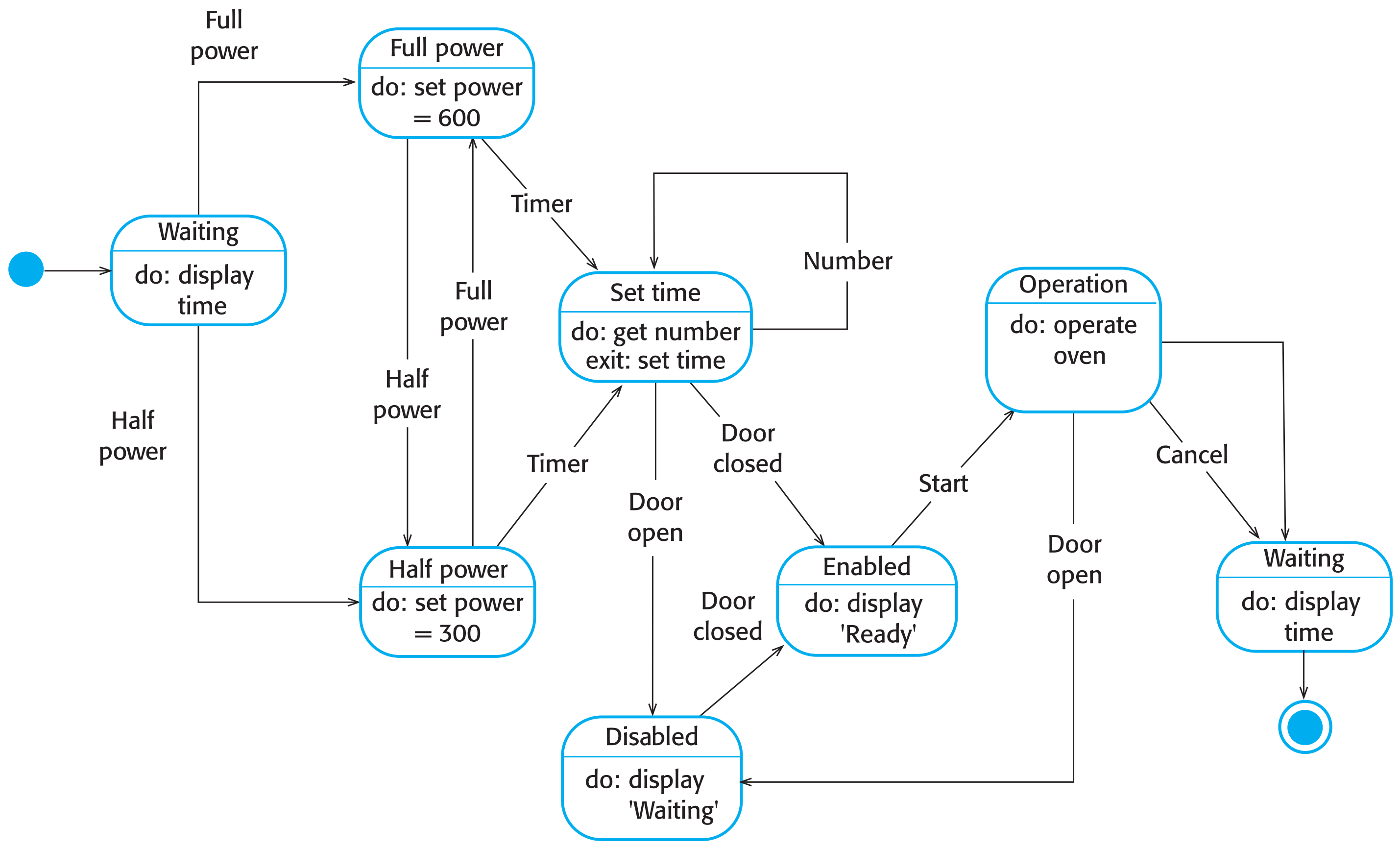
- Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as ‘receiver off hook’ by generating a dial tone.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a **finite number of states and that events (stimuli)** may cause a transition from one state to another.

Behavioral Models: *State Machine Models*

- These model the behaviour of the system in response to external and internal events.
- They show the system’s responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- **Statecharts** are an integral part of the UML and are used to represent state machine models.

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 five 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.



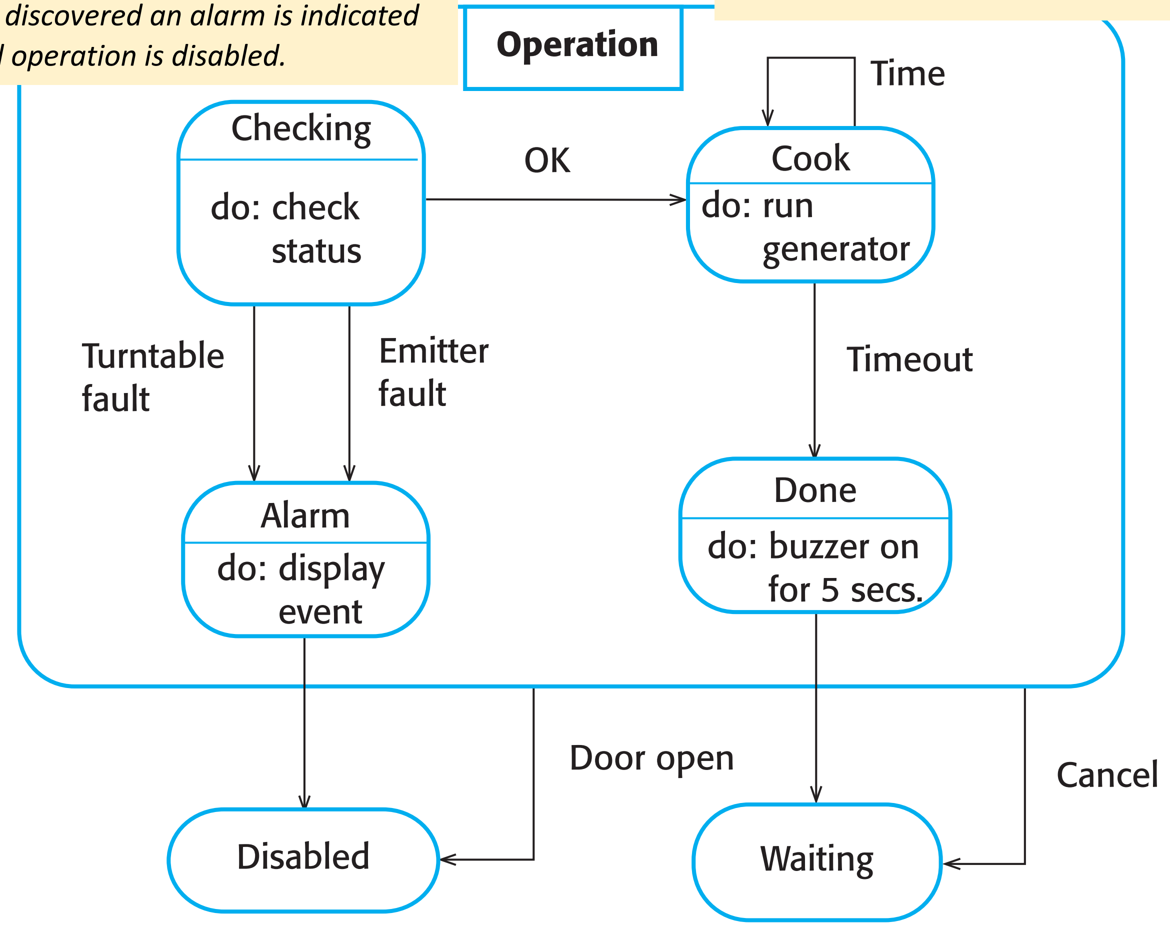
State Diagram of a Microwave Oven

Modeling with Large Number of States

- The problem with state-based modeling is that the number of possible states increases rapidly. For large system models, therefore, you need to hide detail in the models.
- One way to do this is by using the notion of a superstate that encapsulates several separate states. This superstate looks like a single state on a high-level model but is then expanded to show more detail on a separate diagram

includes several sub-states and shows that operation starts with a status check and that if any problems are discovered an alarm is indicated and operation is disabled.

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


Operation
Checking
do: check status
OK
Cook
do: run generator
Time
Turntable fault
Emitter fault
Timeout
Alarm
do: display event
Done
do: buzzer on for 5 secs.
Door open
Cancel
Disabled
Waiting