System Modeling

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 some kind of **graphical notation**, which is now almost always based on notations in the **Unified Modeling Language (UML)**.

System modeling **helps** the analyst to understand the functionality of the system and models are used to communicate with customers. How?

- They help to derive the detailed requirements for a system,
- They describe the system to engineers implementing the system during the design process
- They help in documenting the system's structure and operation.

Importance of system models

- **Importance of modeling in existing systems:** It is useful in 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.
- **Importance of modeling in new systems:** It is useful 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.
- It is possible to **generate** a complete or partial system **implementation** from the system model.
- A system model is not a complete representation of system. It purposely leaves out detail to make it **easier to understand** but it describes out the **most salient characteristics**.

System perspectives

- An external 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.

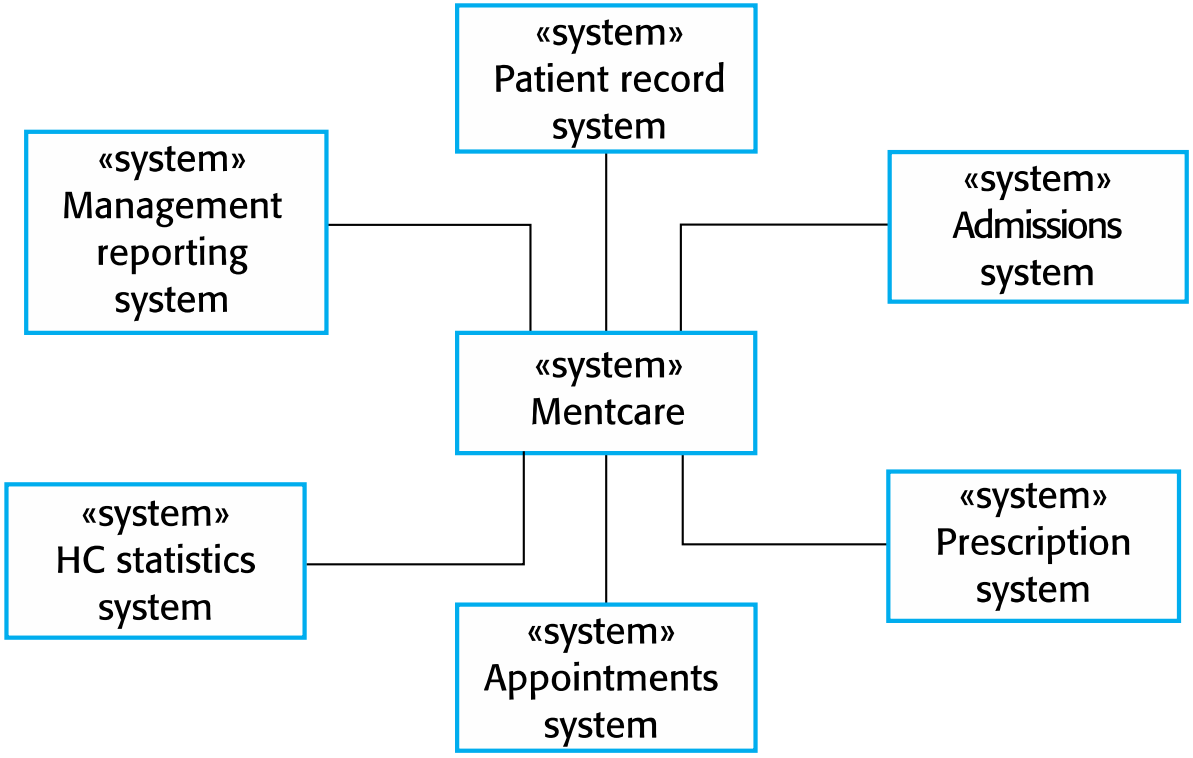
UML diagram types

- Activity diagrams, which show the activities involved in a process or in data processing .
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, which show the object classes in the system and the associations between these classes.
- State diagrams, which show how the system reacts to internal and external events.

Context models

- **Early stage of requirement specification: Deciding system boundaries**, that is, on what is and is not part of the system being developed. It involves working with system stakeholders to limit the system costs and the time needed like automated or manual; re-use or implement; and so on...
- **In some cases, the boundary is clear** like, where an automated system is replacing an existing manual. There is **difficult to decide the boundary** in other cases. For example, maintaining database. The advantage of relying on other systems for information is that you avoid duplicating data. The major disadvantage, however, is that using other systems may make it slower to access information, and if these systems are unavailable, then it may be impossible to use your system.
- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Architectural models show the system and its relationship with other systems.

The context of the Mentcare system



Process perspective

Context models simply show the other systems in the environment, not how the system being developed is used in that environment.

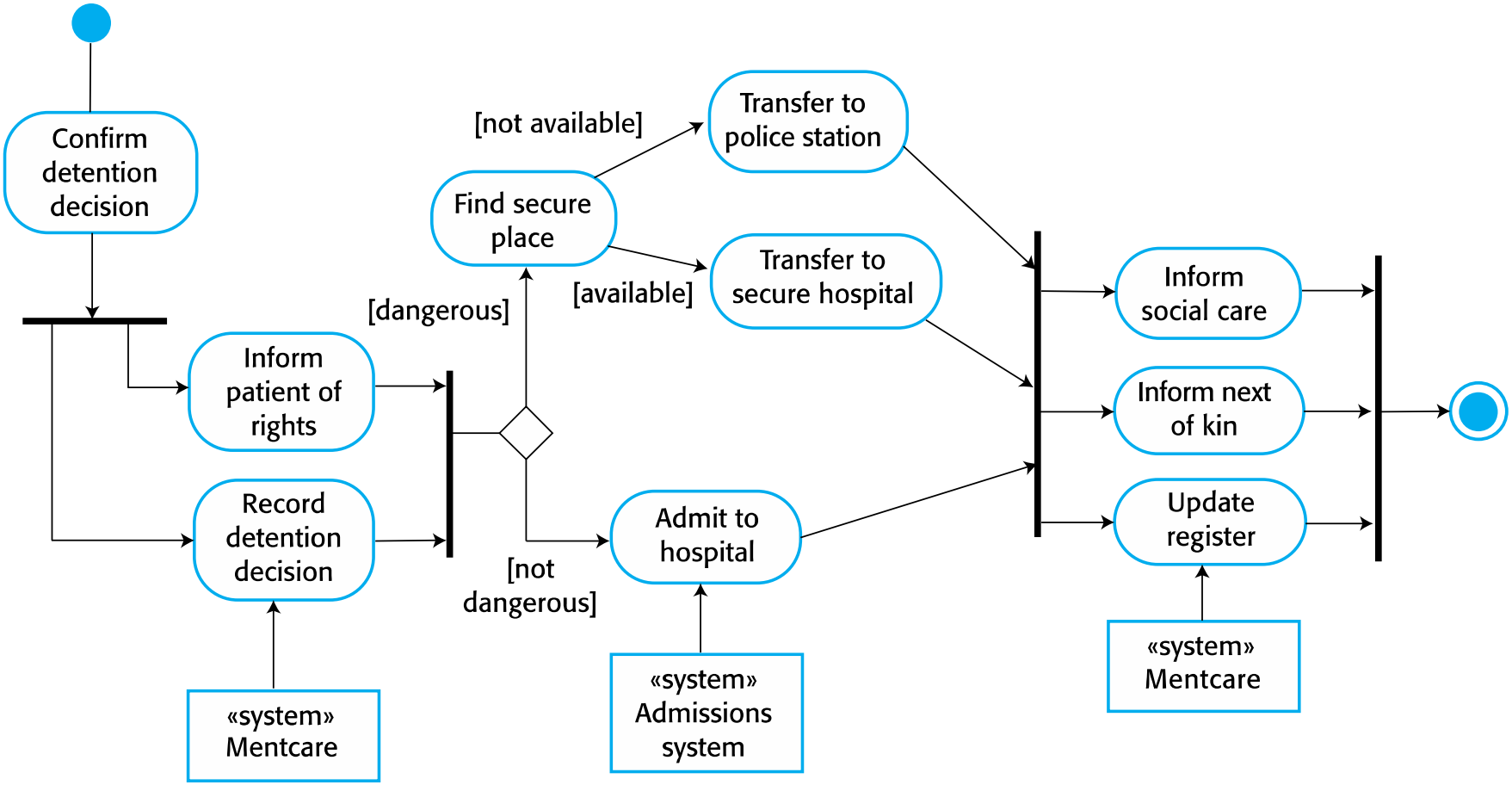
- External systems might produce data for or consume data from the system. They might share data with the system, or they might be connected directly, through a network or not connected at all. They might be physically co-located or located in separate buildings. These relations may affect the requirements and system design thus, simple context models are augmented with other models, describing how the systems is used.

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.

Process model of involuntary detention

- It shows the activities in a process and the flow of control from one activity to another.
- The start and end are indicated by a filled circle and filled circle inside another circle respectively.
- Round corners rectangles represent activities and arrows represent flow of work between them.
- Solid bar indicates activity coordination. When the flow from a solid bar leads to a number of activities, these may be executed in parallel.



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 modeling.

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 diagramatically to provide an overview of the use case and in a more detailed textual form.

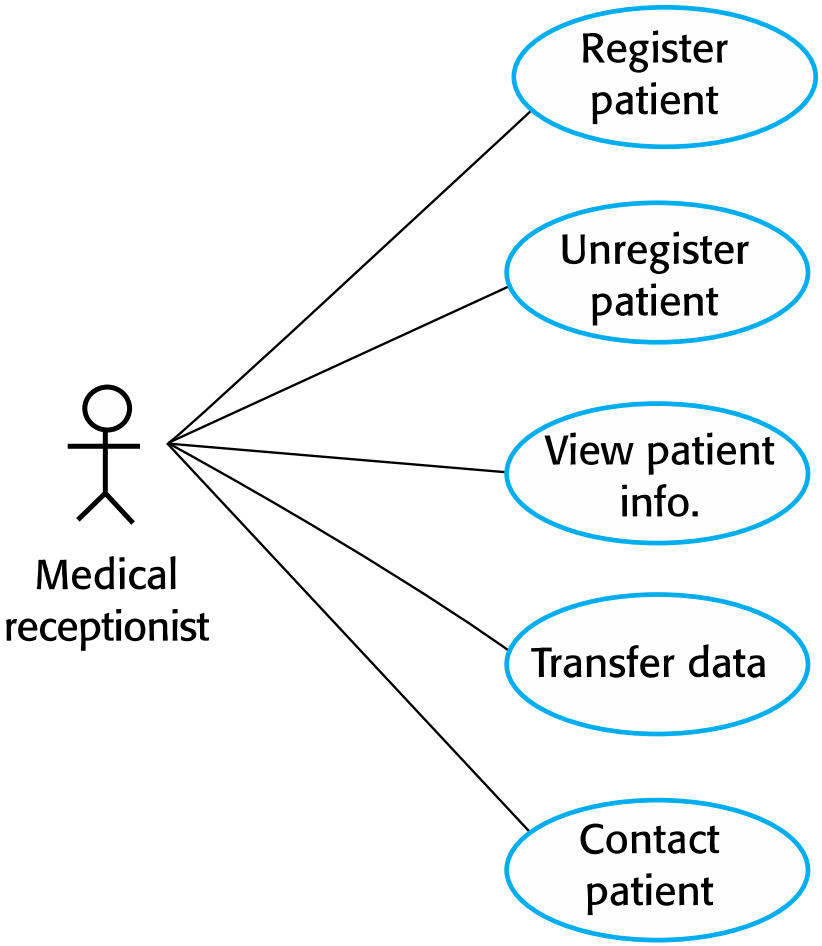
Transfer-data use case

- A use case can be taken as a simple description of what a user expects from a system in that interaction.
- Example, use case from the Mentcare system that represents the task of uploading data from the Mentcare system to a more general patient record system.
- It contain two actors: the operator who is transferring the data and the patient record system. Note, actor can also represent other external systems and hardware apart from user.
- Use case diagrams give a simple overview of an interaction, and more detail can be added for complete interaction description. This detail can either be a simple textual description, a structured description in a table, or a sequence diagram.

Tabular description of the ‘Transfer data’ use-case

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the Mentcase system 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 in the Mentcare system involving the role ‘Medical Receptionist’

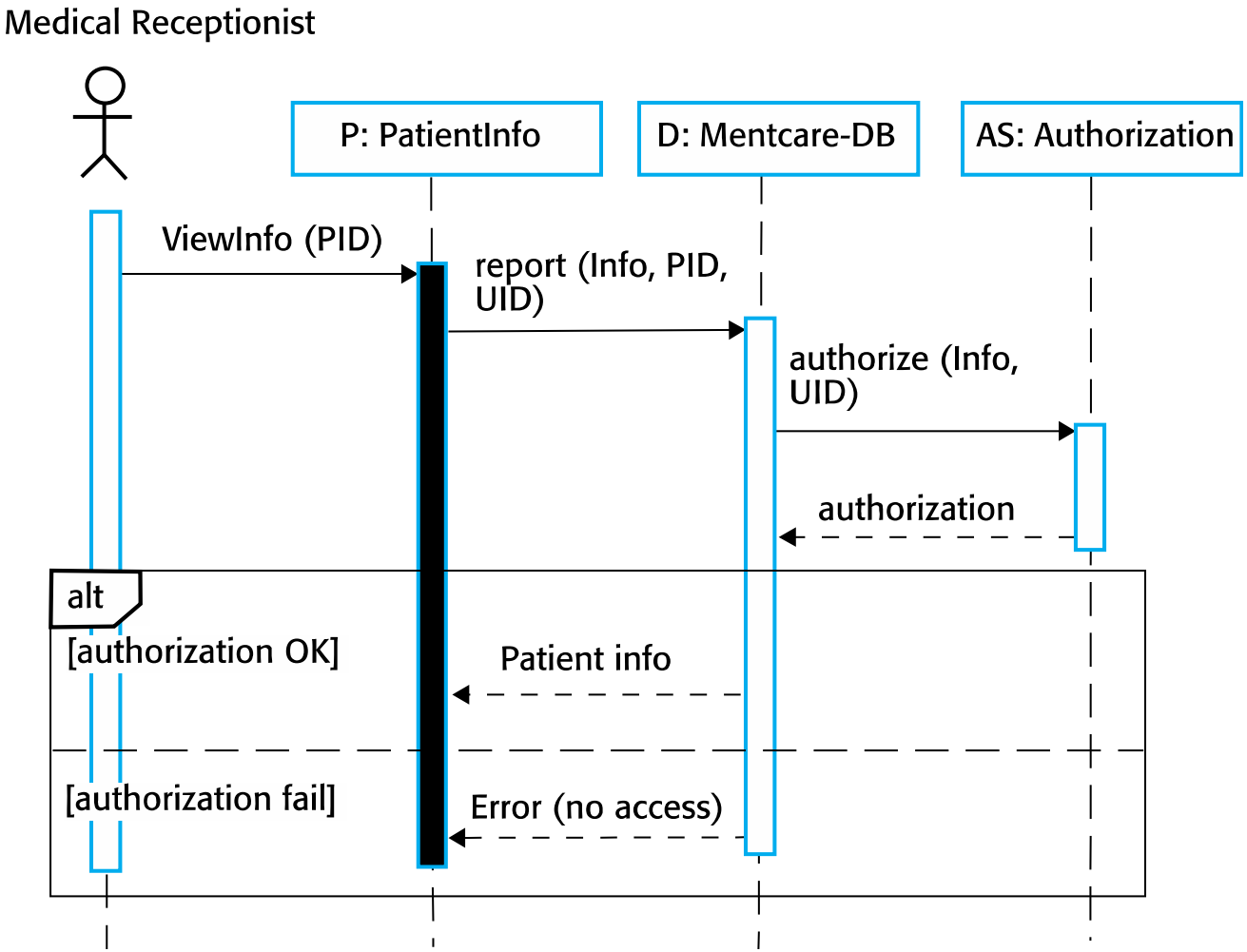


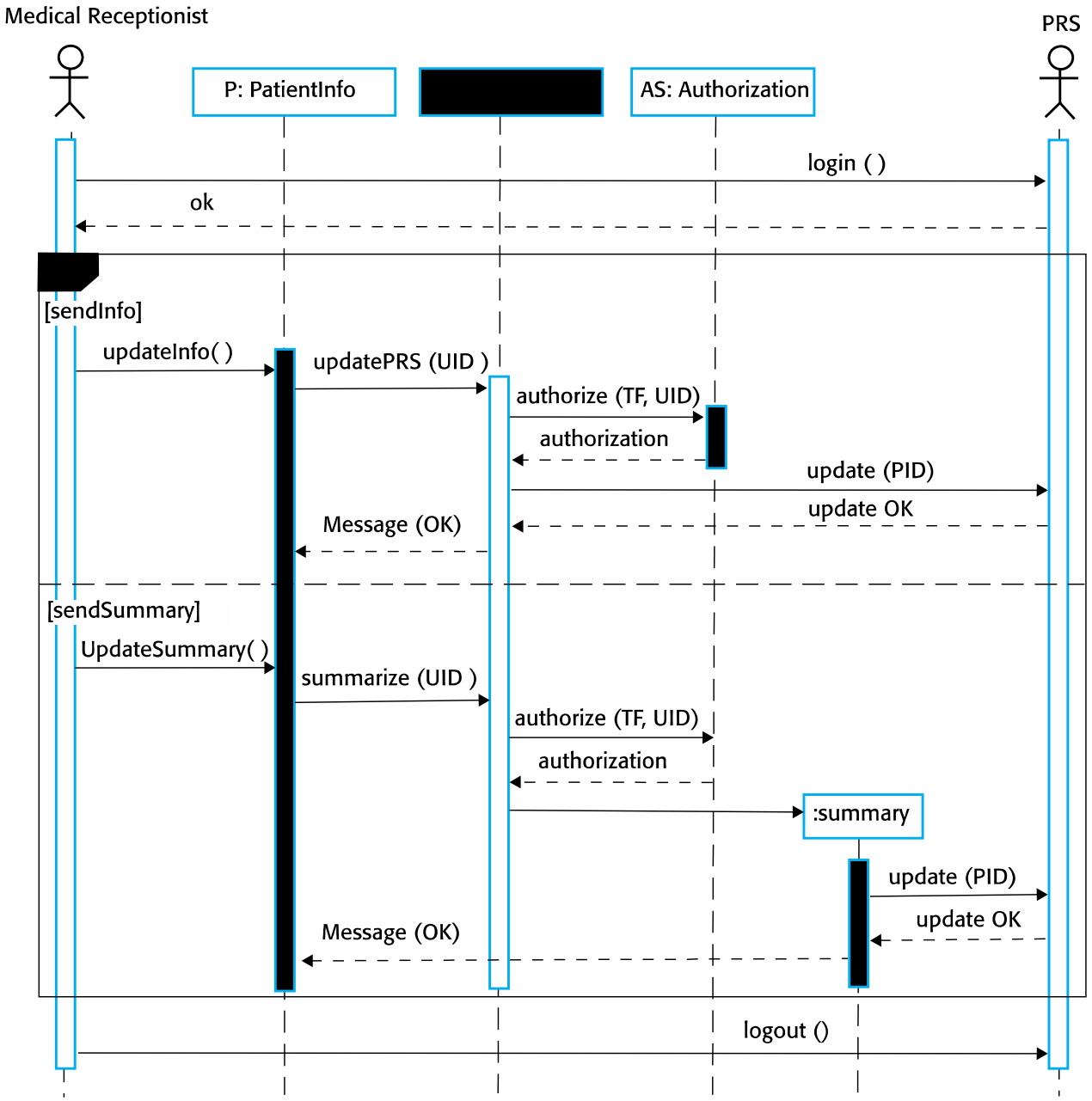
Composite use case diagrams show a number of different use cases. Sometimes it may not be impossible because of the number of use cases. In such cases, you may develop several diagrams, each of which shows related use cases.

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.
- The rectangle on the dotted lines indicates the lifeline of the object concerned.
- You read the sequence of interactions from top to bottom.
- A box named alt is used with the conditions indicated in square brackets, with alternative interaction options separated by a dotted line.

Sequence diagram for View patient information





Sequence diagram for Transfer Data

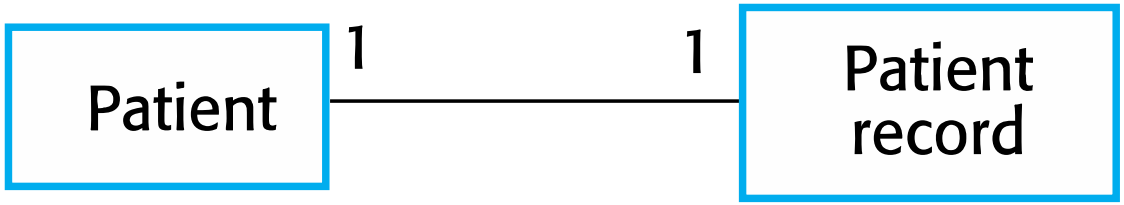
Structural models

- 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.

Class diagrams

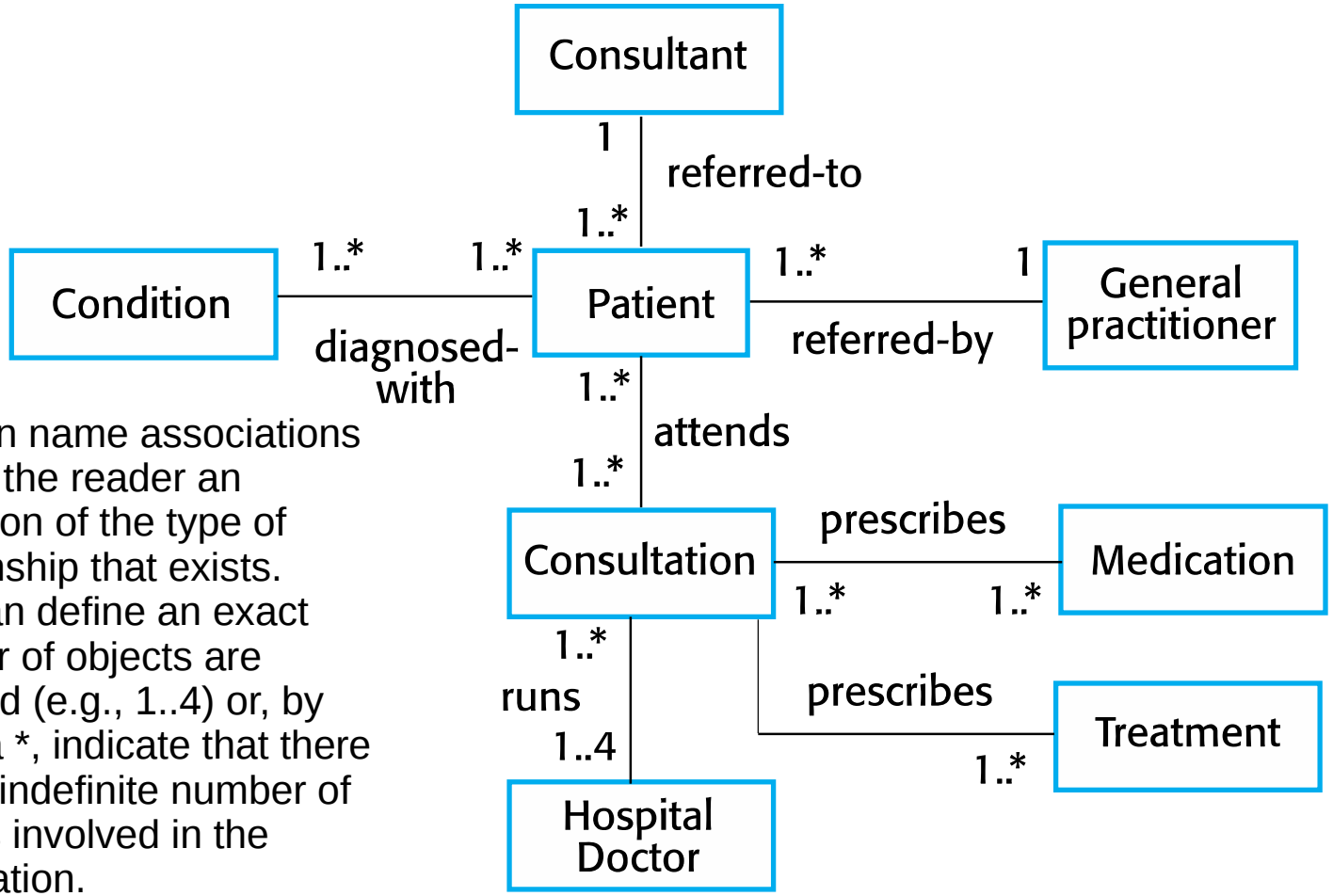
- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that 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.
- When developing a model, the first stage is usually to look at the world, identify the essential objects, and represent these as classes.
- At this stage, you do not need to say what the association is.

UML classes and association



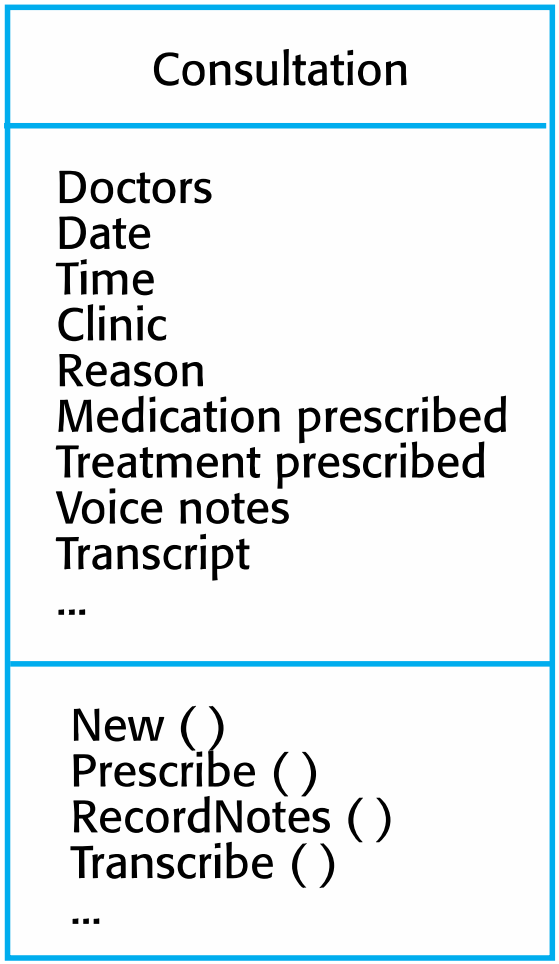
Association annotated with a 1, means that there is a 1:1 relationship.

Classes and associations in the MHC-PMS



You can name associations to give the reader an indication of the type of relationship that exists. One can define an exact number of objects are involved (e.g., 1..4) or, by using a *, indicate that there are an indefinite number of objects involved in the association.

The Consultation class



- Class diagrams look like semantic data models containing data entities, their associated attributes, and the relations between these entities.
- When showing the associations between classes, it is best to represent these classes in the simplest possible way, without attributes or operations.
- To define objects in more detail, one can add information about their attributes (the object's characteristics) and operations (the object's functions). For example, a Patient object has the attribute Address, and you may include an operation called ChangeAddress, which is called when a patient indicates that he or she has moved from one address to another. It contains:
  1) The name of the object class is in the top section.
  2) The class attributes are in the middle section.
  3) The operations associated with the object class are in the lower section of the rectangle.

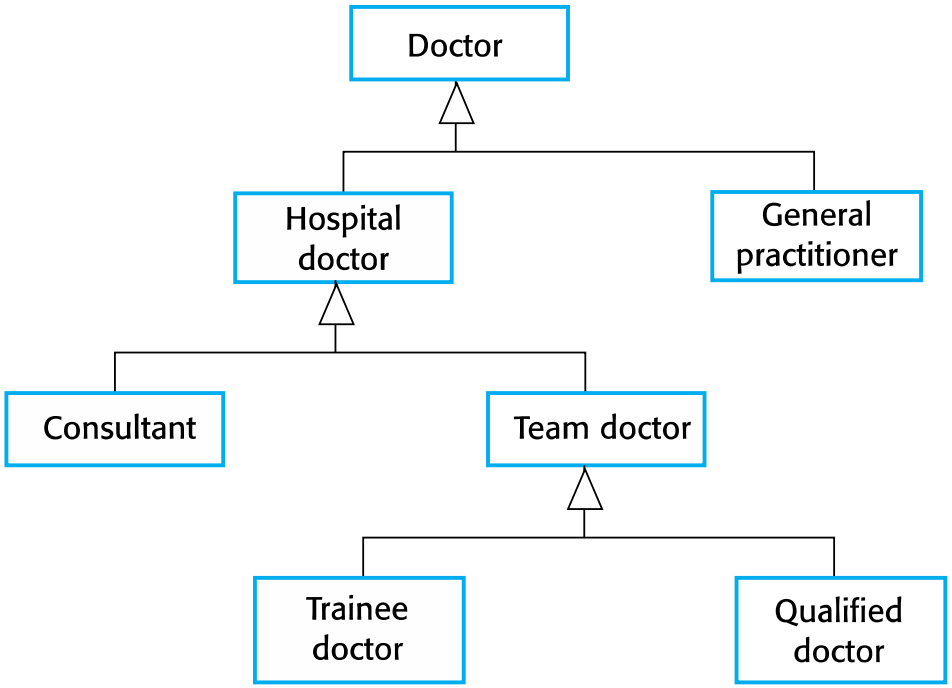
Generalization

- 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 e.g. squirrels and rats are rodents and so share the characteristics of rodents like all rodents have teeth for gnawing

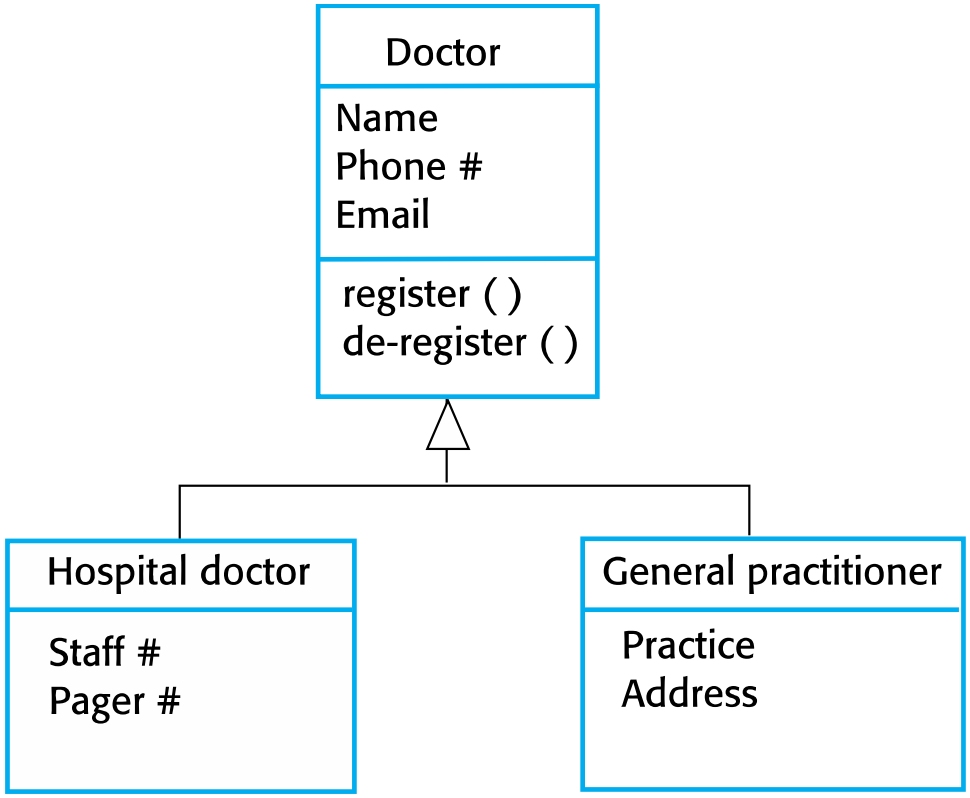
Generalization

- 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 superclasses. These lower-level classes then add more specific attributes and operations.

A generalization hierarchy

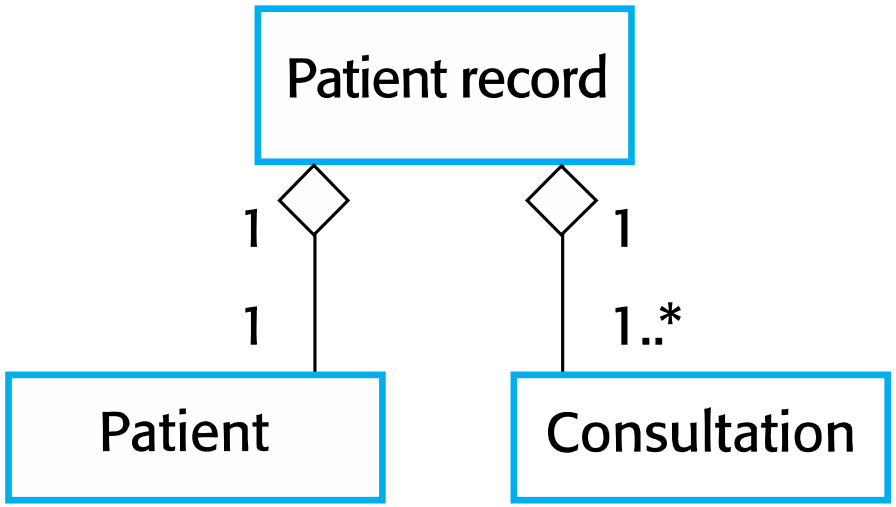


A generalization hierarchy with added detail



Object class aggregation models

- Objects in the real world are often made up of different parts. For example, a course may be composed of a book, slides, quizzes, and recommendations for further reading.
- The UML provides a special type of association between classes called aggregation, which means that one object (the whole) is composed of other objects (the parts).
- To define aggregation, a diamond shape is added to the link next to the class that represents the whole.



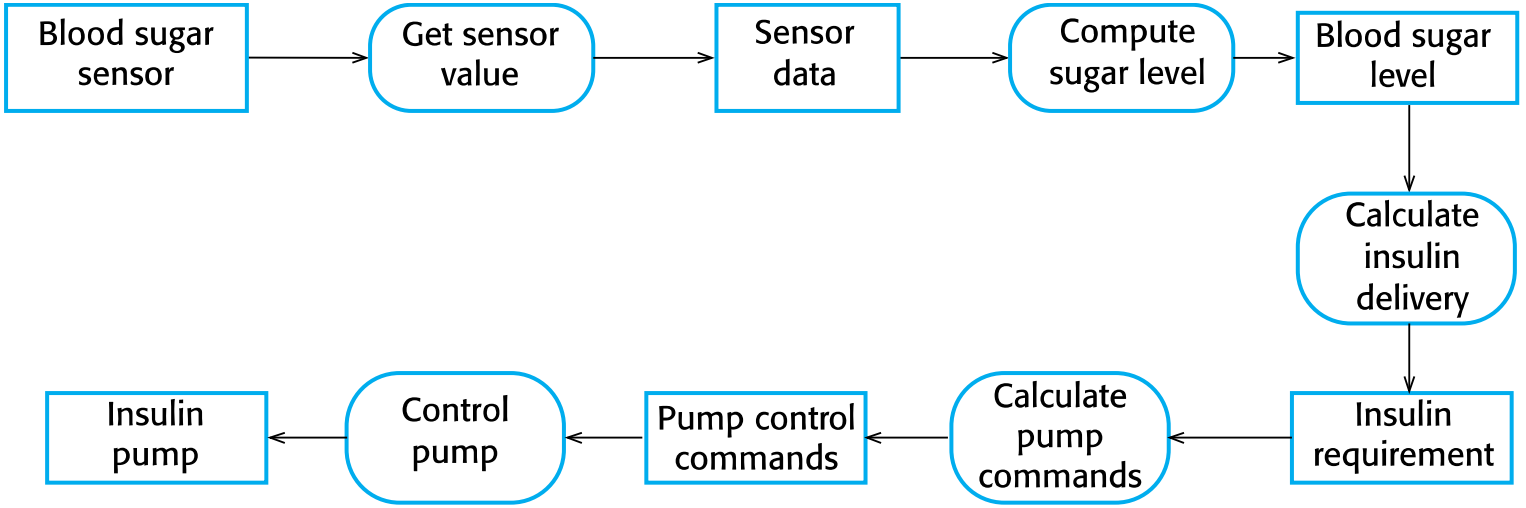
Behavioral models

- 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.

Data-driven modeling

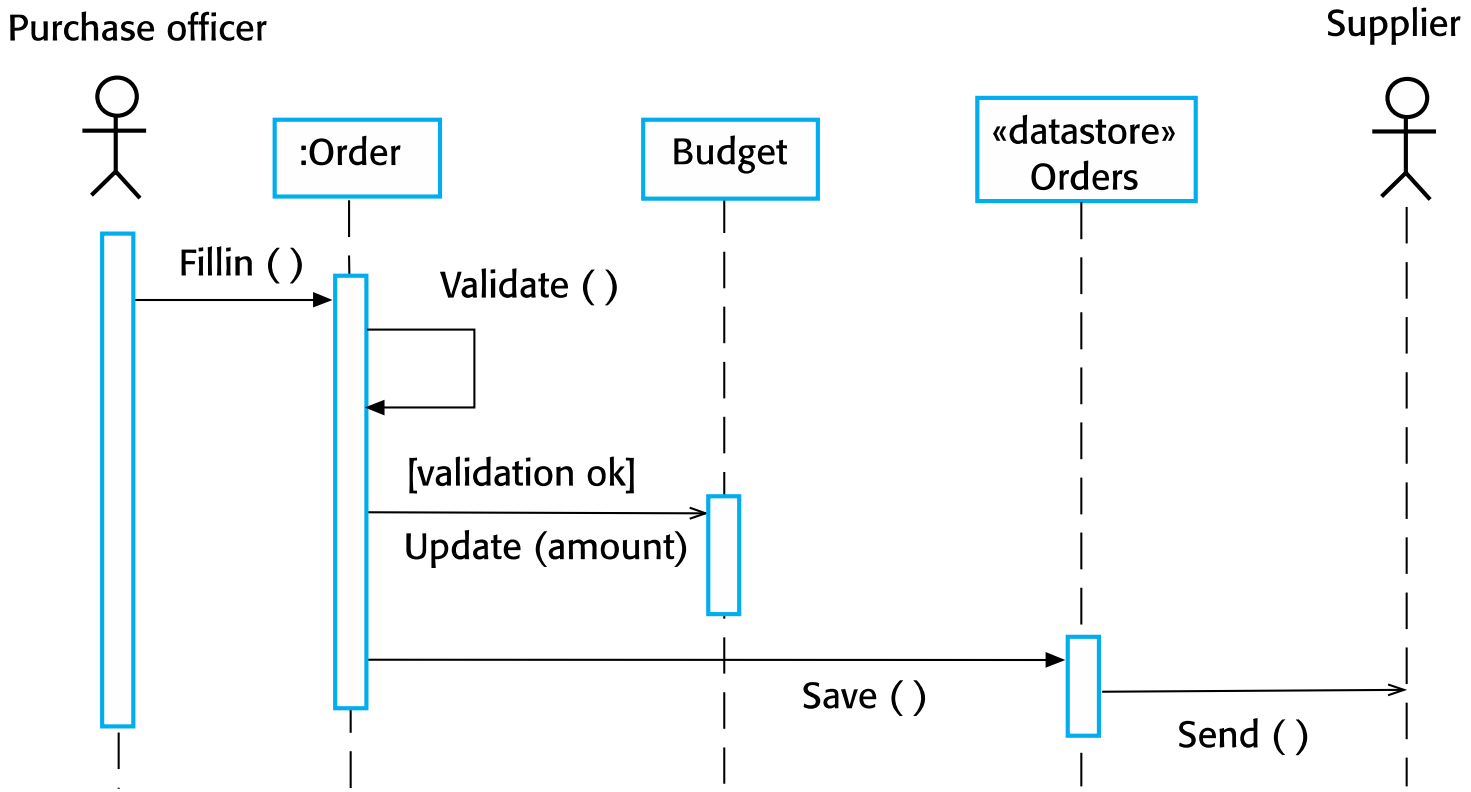
- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- For example, a phone billing system will accept information about calls made by a customer, calculate the costs of these calls, and generate a bill for that customer.
- 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.

An activity model of the insulin pump's operation



- Data-flow diagram (DFD) can be used to illustrate processing steps in a system.
- They are useful because:
  - Tracking and documenting how data associated with a particular process moves through the system help analysts and designers understand what is going on in the process.
  - They are simple and intuitive to stakeholders.
- They can be represented in the UML using the activity diagram.
  - Processing steps, represented as activities (rounded rectangles)
  - Data flowing between these steps, represented as objects (rectangles).

Order processing



- Alternative way is UML sequence diagrams. If it is drawn such that messages are only sent from left to right, then they show the sequential data processing.
- Sequence models highlight objects in a system, whereas data-flow diagrams highlight the operations or activities. In practice, nonexperts seem to find data-flow diagrams more intuitive, but engineers prefer sequence diagrams.

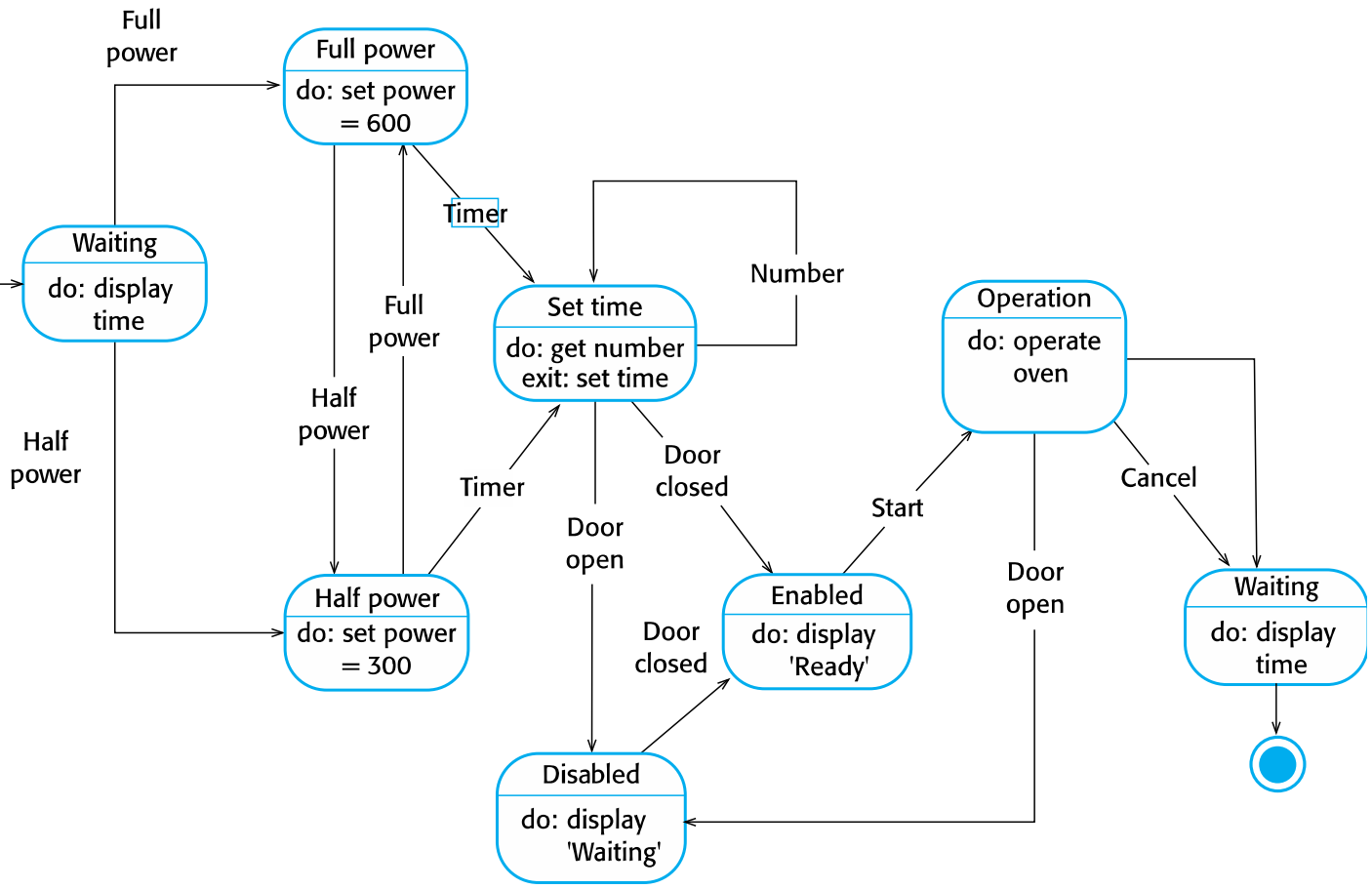
Event-driven modeling

- 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. For example, a system controlling a valve may move from a state “Valve open” to a state “Valve closed” when an operator command (the stimulus) is received.

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.
- UML State diagrams show system states and events that cause transitions from one state (rounded rectangles) to another. They do not show the flow of data within the system.

State diagram of a microwave oven



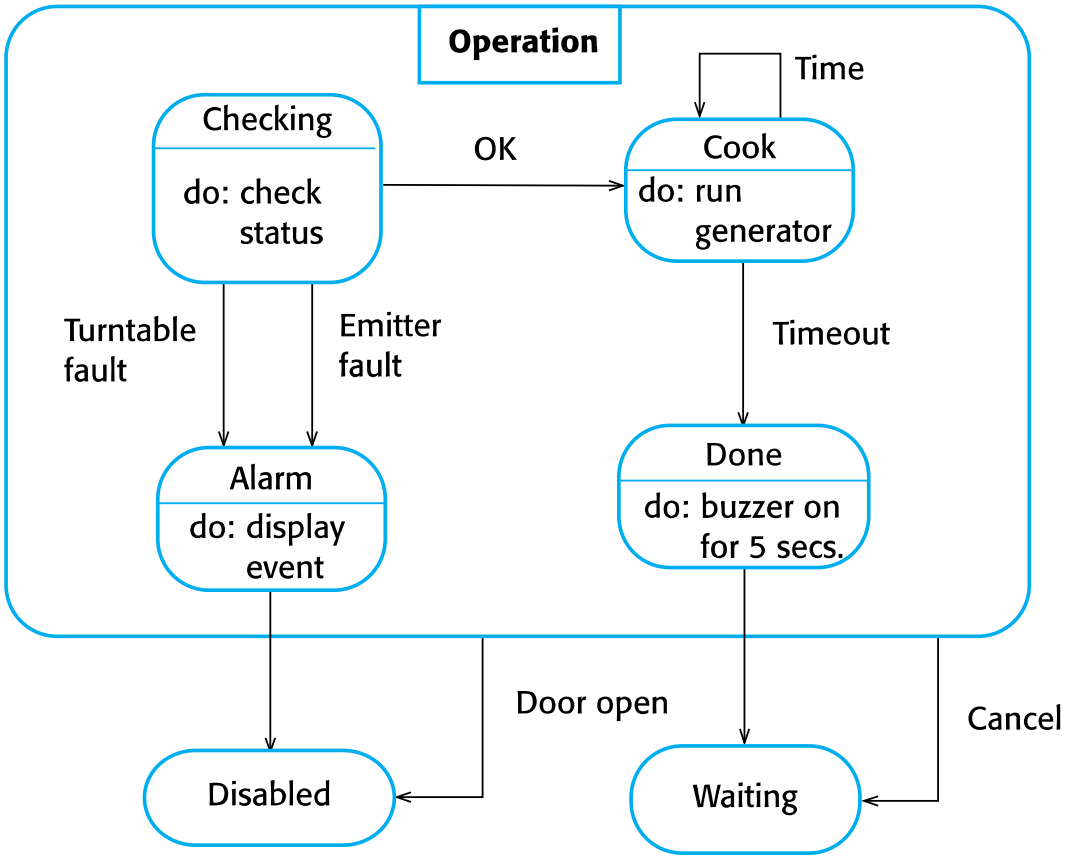
Rounded rectangles depicting states, may provide a brief description (following “do”) of the actions taken in that state.

The labeled arrows represent stimuli that force the transition.

Start and end states can be depicted just like we did in activity diagrams.

Microwave oven operation

- Problem with state-based modeling: the number of possible states increases rapidly.
- Thus, we hide detail in the models using the notion of a “superstate”.
- Superstate looks like a single state on a high-level model but is then expanded to show more detail on a separate diagram.



States and stimuli for the microwave oven (a)

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.

- State models provide overview of event processing.
- More details can be added using a table to list the states and events that stimulate state transitions along with a description of each state and event.

States and stimuli for the microwave oven (b)

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.