Module-3
System Modeling

Vaishnavi
Assistant Professor

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 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.

✧ In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

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.

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 suppon 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
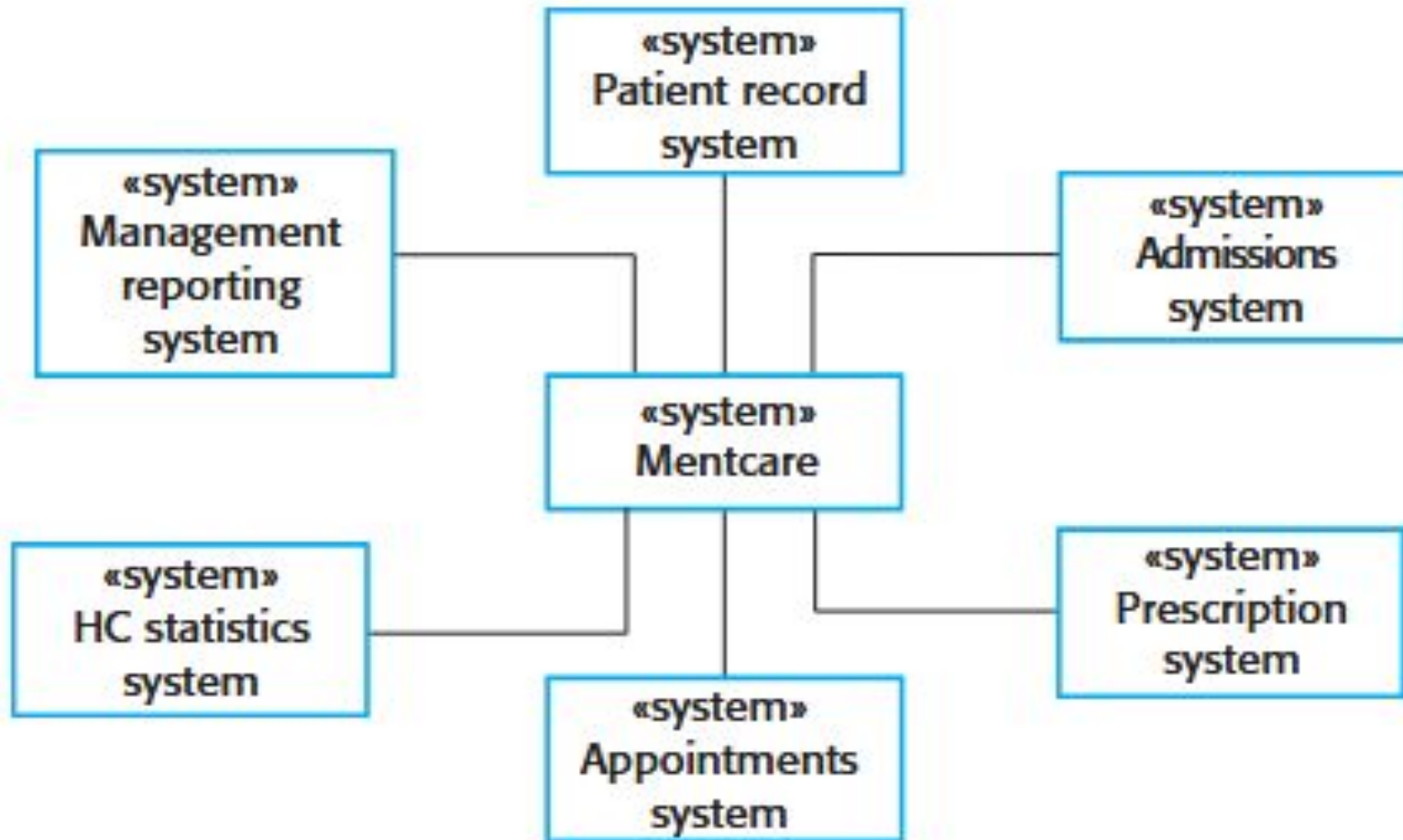
  - Models have to be both correct and complete.

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 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 eñect on the system requirements.

❑ Defining a system boundary is a political judgment

  • There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

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.

❏   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.

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.
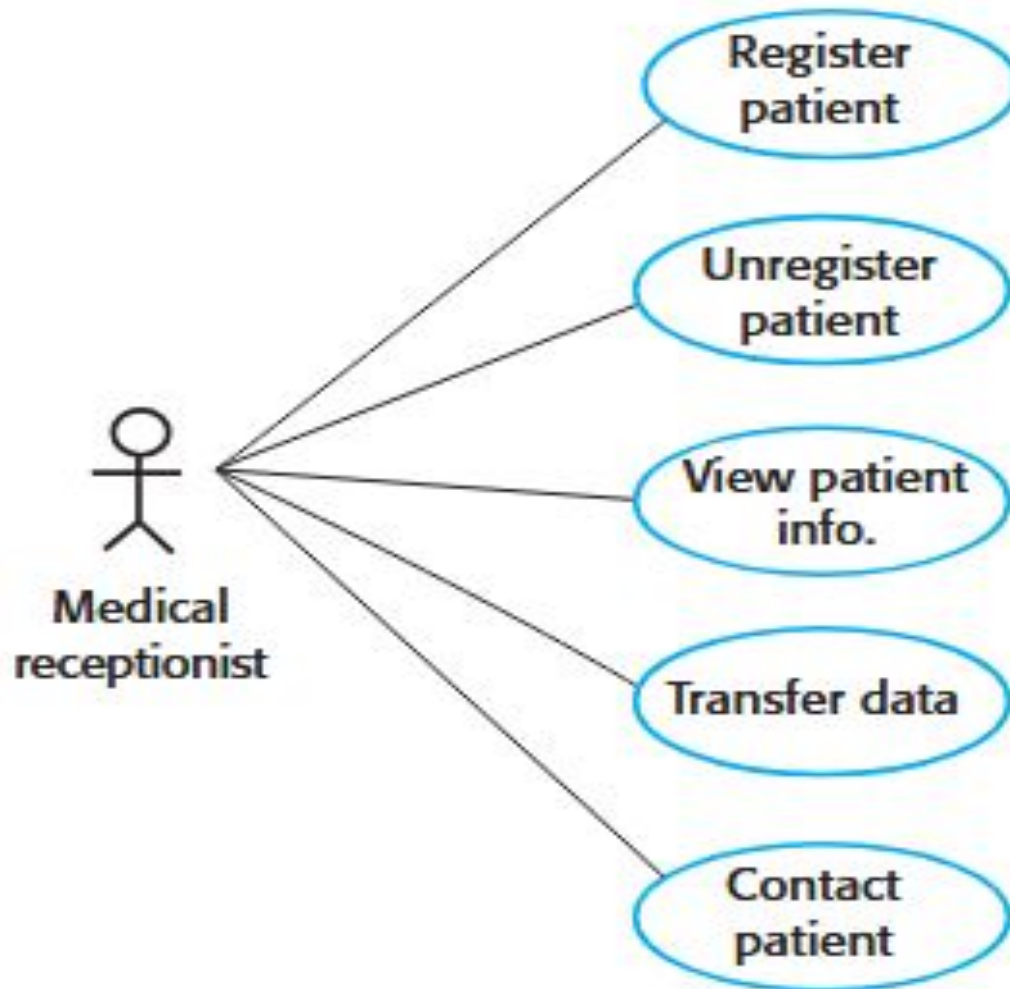
Transfer-data use case

Arrow direction: Medical receptionist initiates the transaction and data is transferred to the patient record system

Tabular description of the 'Transfer data' use-case

Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare 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.

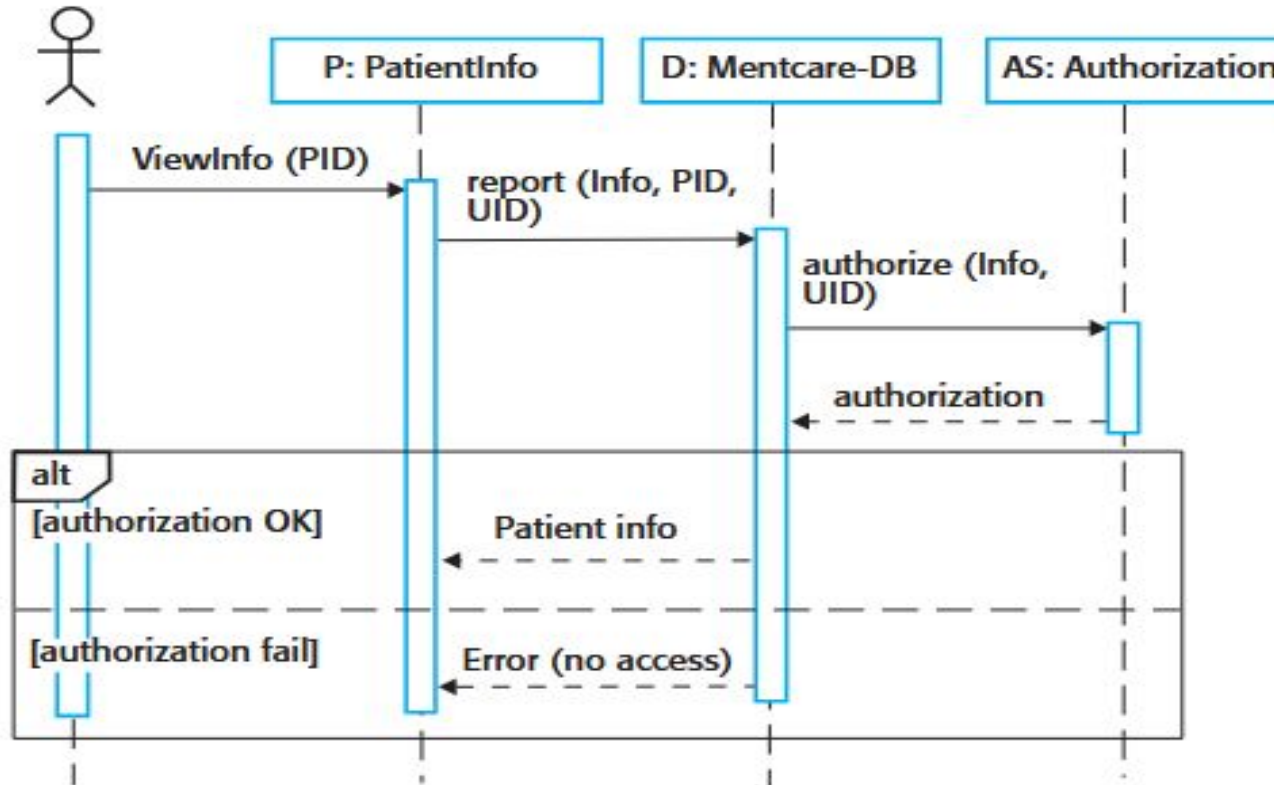# Use cases in the MHC-PMS involving the role 'Medical Receptionist'

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

Medical Receptionist

P: PatientInfo

D: Mentcare-DB

AS: Authorization

ViewInfo (PID)

report (Info, PID, UID)

authorize (Info, UID)

authorization

alt

[authorization OK]

Patient info

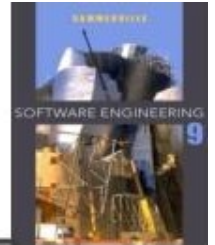[authorization fail]

Error (no access)

The medical receptionist triggers the ViewInfo method in an
1. instance P of the PatientInfo object class, supplying the patient's identifier, PID. P is a user interface object, which is displayed as a form showing patient information.
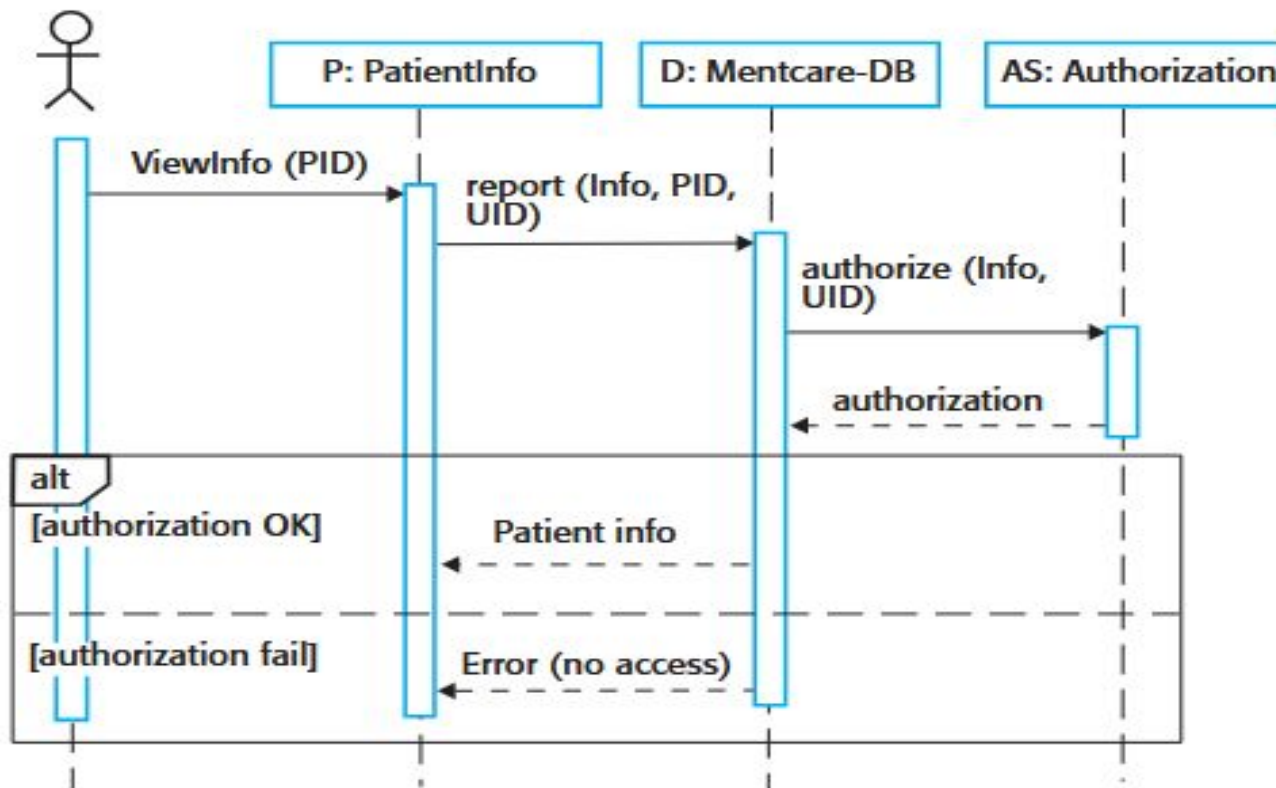
2.   The instance P calls the
    database to return the information required, supplying the receptionist's identifier to allow security checking (at this stage, we do not care where this UID comes from).

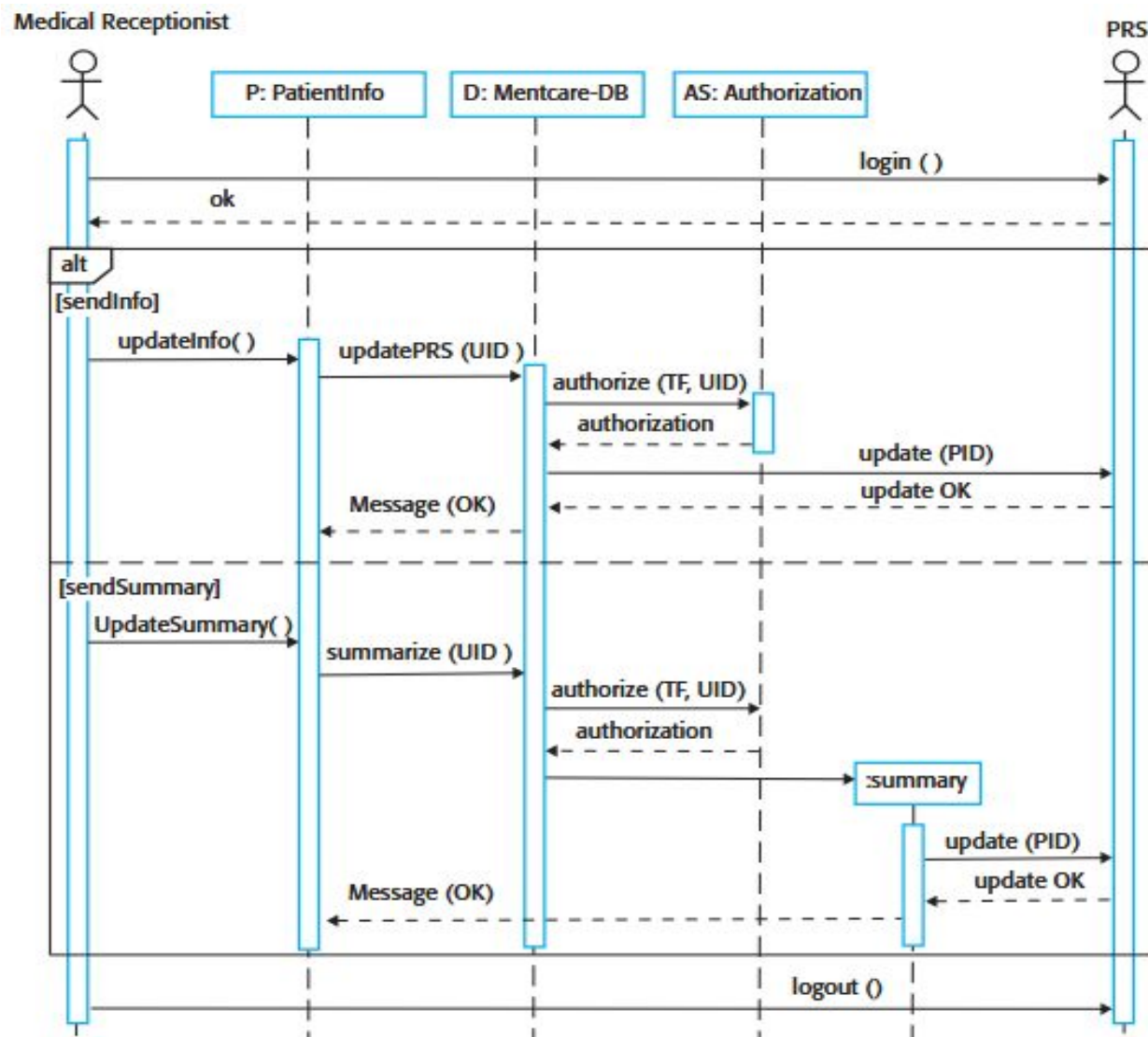# Sequence diagram for View patient information

Medical Receptionist

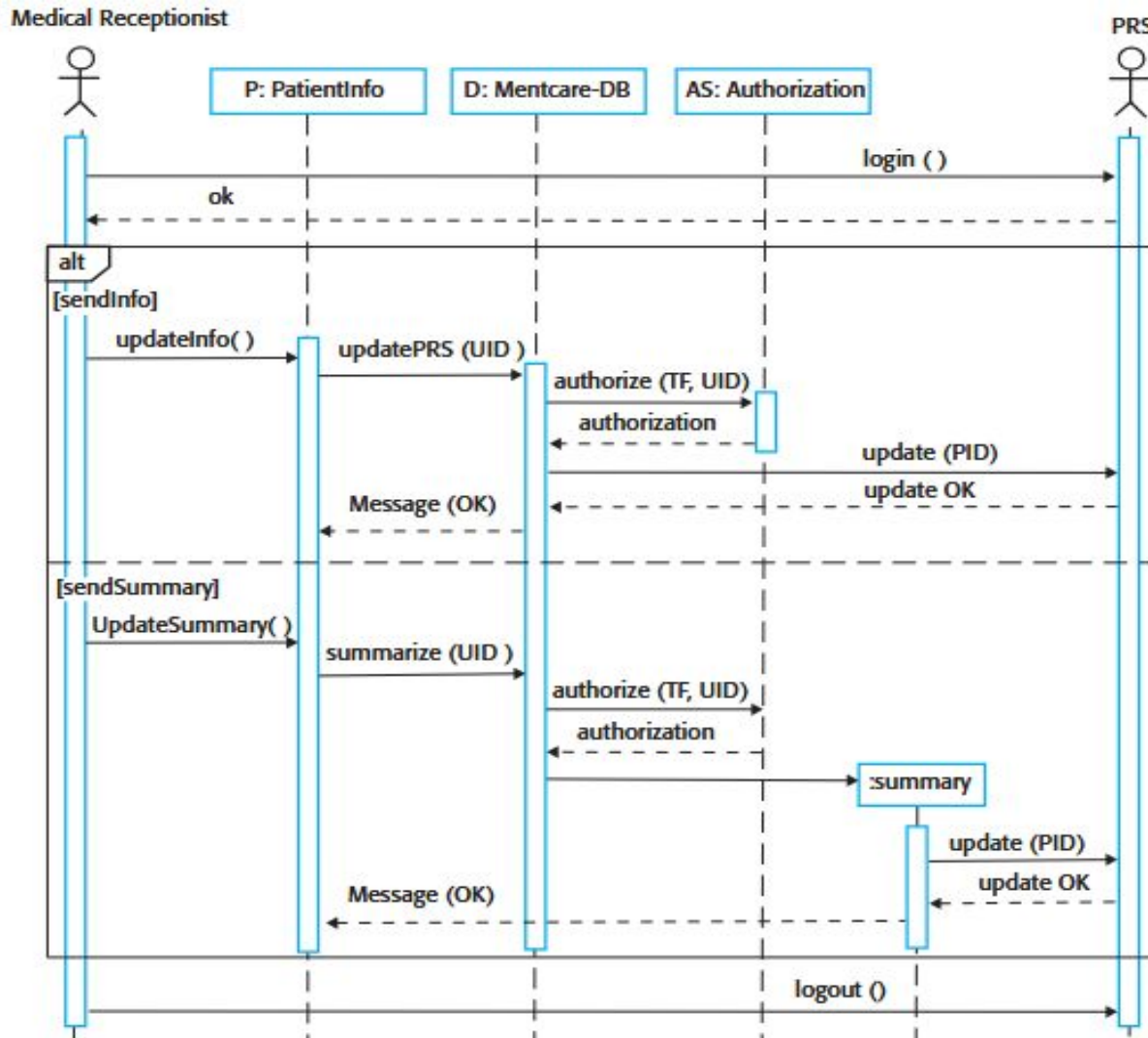The database checks with an authorization system that the user is authorized for this action.

If authorized, the patient information is returned and a form on the user's screen is filled in. If authorization fails, then an error message is returned.

Sequence diagram for Transfer Data

1 The receptionist logs on to the PRS.

2. There are two options available. These allow the direct transfer of updated patient information to the PRS and the transfer of summary health data from the MHC-PMS to the PRS.

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

Sequence diagram for Transfer Data

4. Personal information may be transferfed difectly from the user interface object to the Patient Record System (PRS). Alternatively, a summary record may be created from the database and that record is then tfansferred.

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

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.
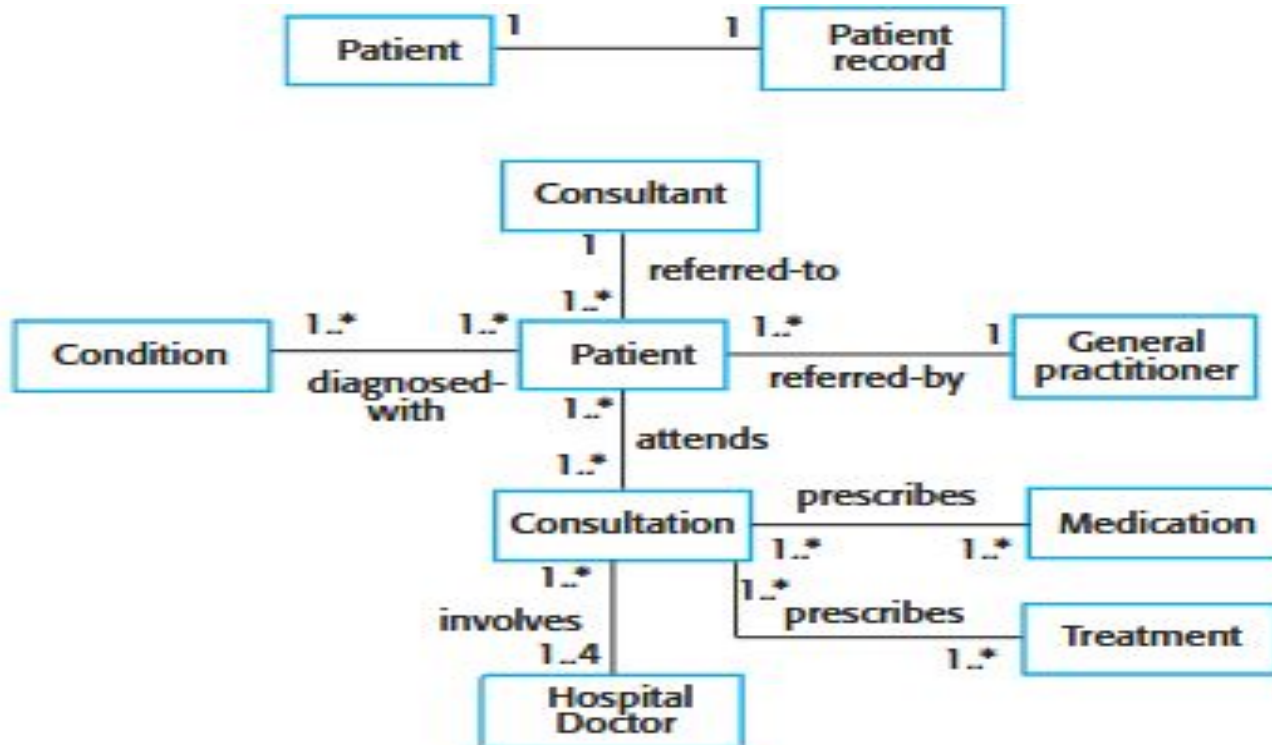
Class diagrams

❑ 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.

UML classes and association

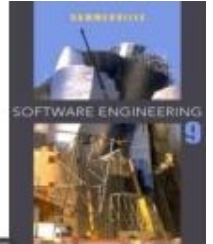|	
Patient	Patient record

1 ——————— 1

- Shows the relationship between Patient and Patient Record
    - This shows 1 to 1 relationship between the two, each patient has one record, and there is only one patient for each patient record.

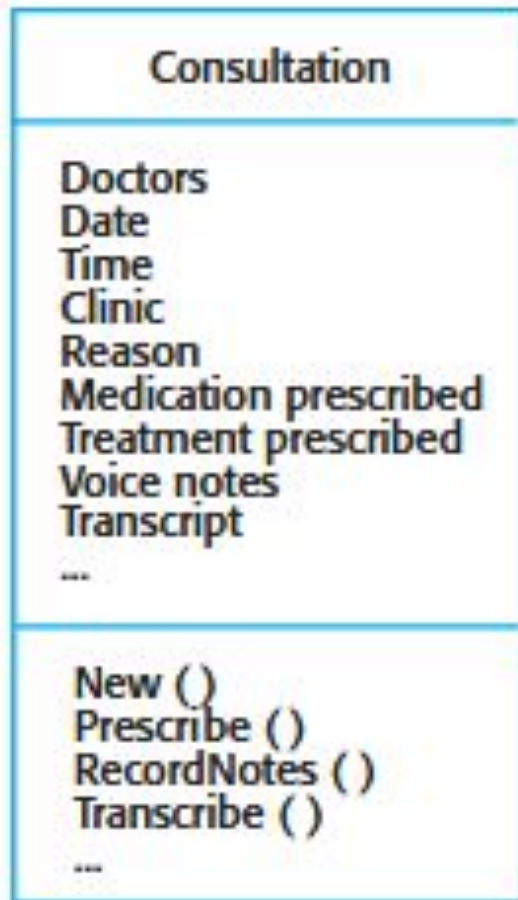# Classes and associations in the Mentcare system

The '*' is used to represent many. This diagram shows a one to many relationship between Patient and Consultation, i.e., many patients can have many consultations and many consultations can have many patients.

The Consultation class

UML diagram for class
Consultation

Name of object
class

Class attributes: name
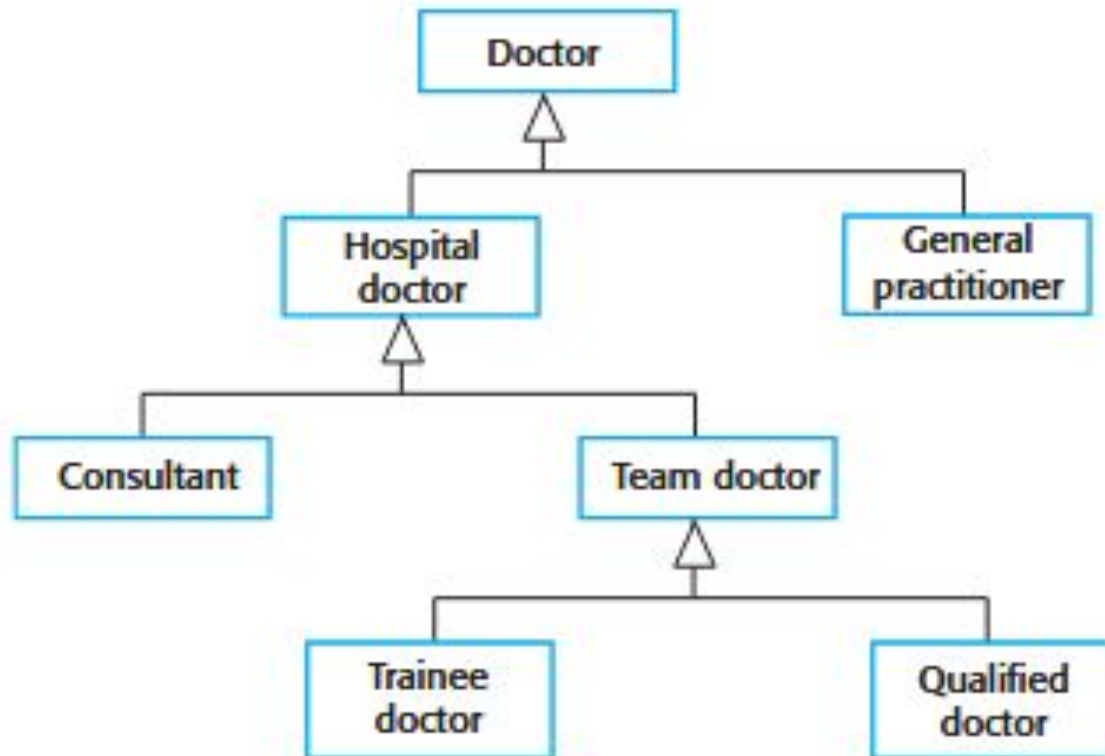required,
optional to include types

Operations or
methods

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 dilerent members of these classes have some common characteristics e.g. squirrels and rats are rodents.
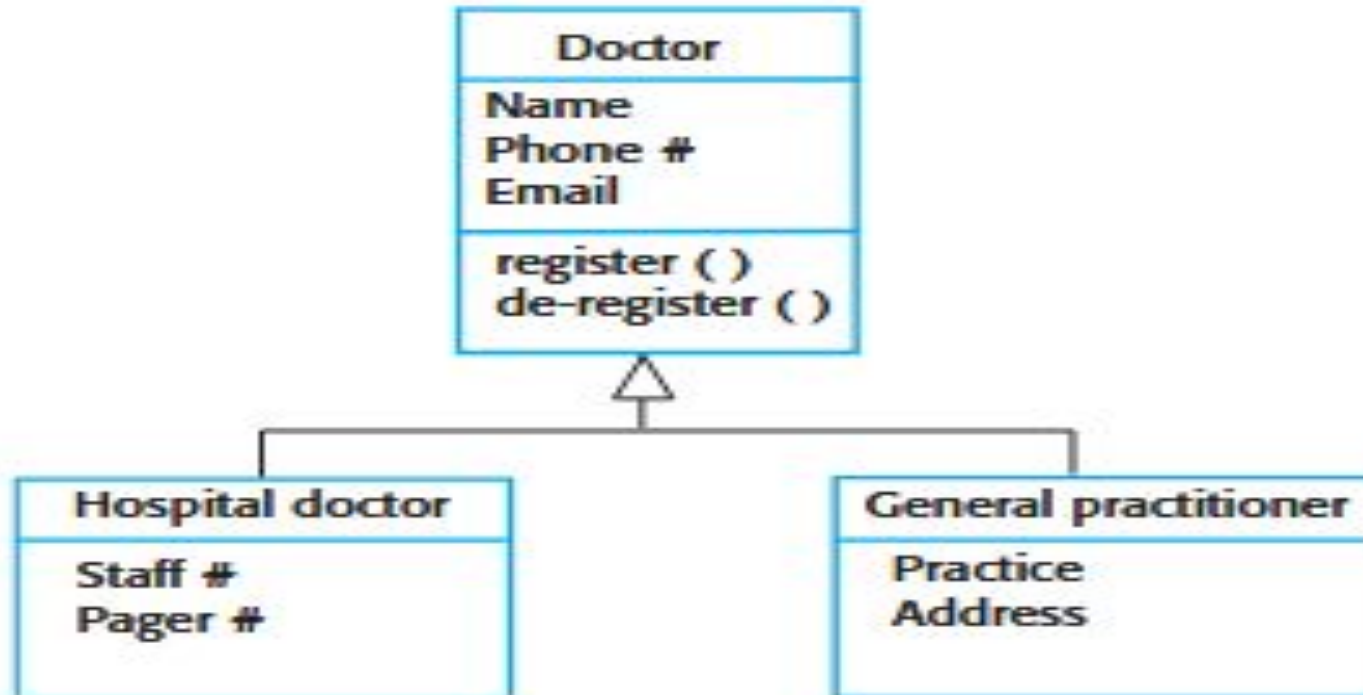
Generalization

❏ 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 añected 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 or subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.
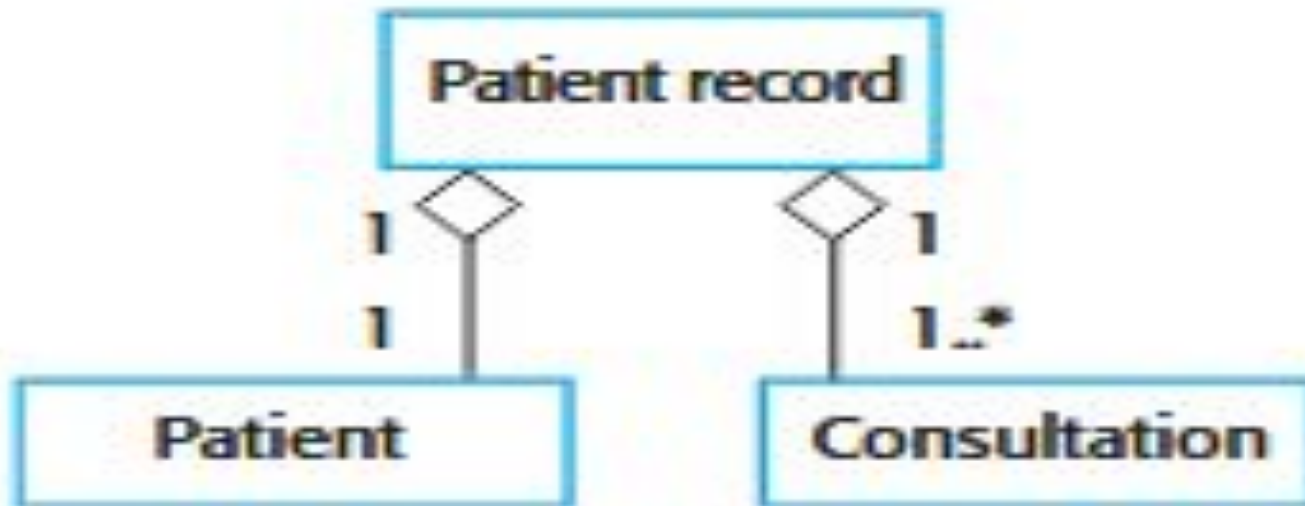
s?

A generalization hierarchy

A generalization hierarchy with added detail

Object class aggregation models

❏   An aggregation model shows how classes that are collections are composed of other classes.

❏   Aggregation models are similar to the part-of relationship in semantic data models.
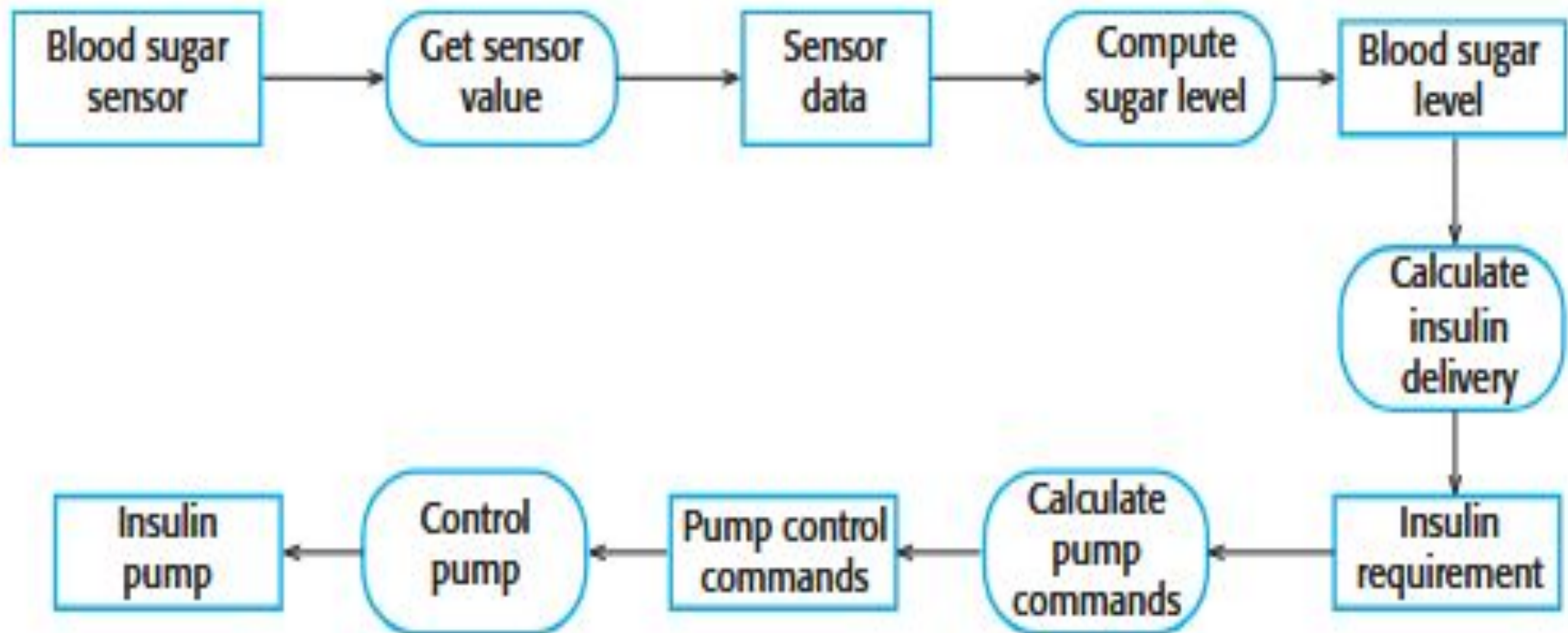
The aggregation association

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.

❑ 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.
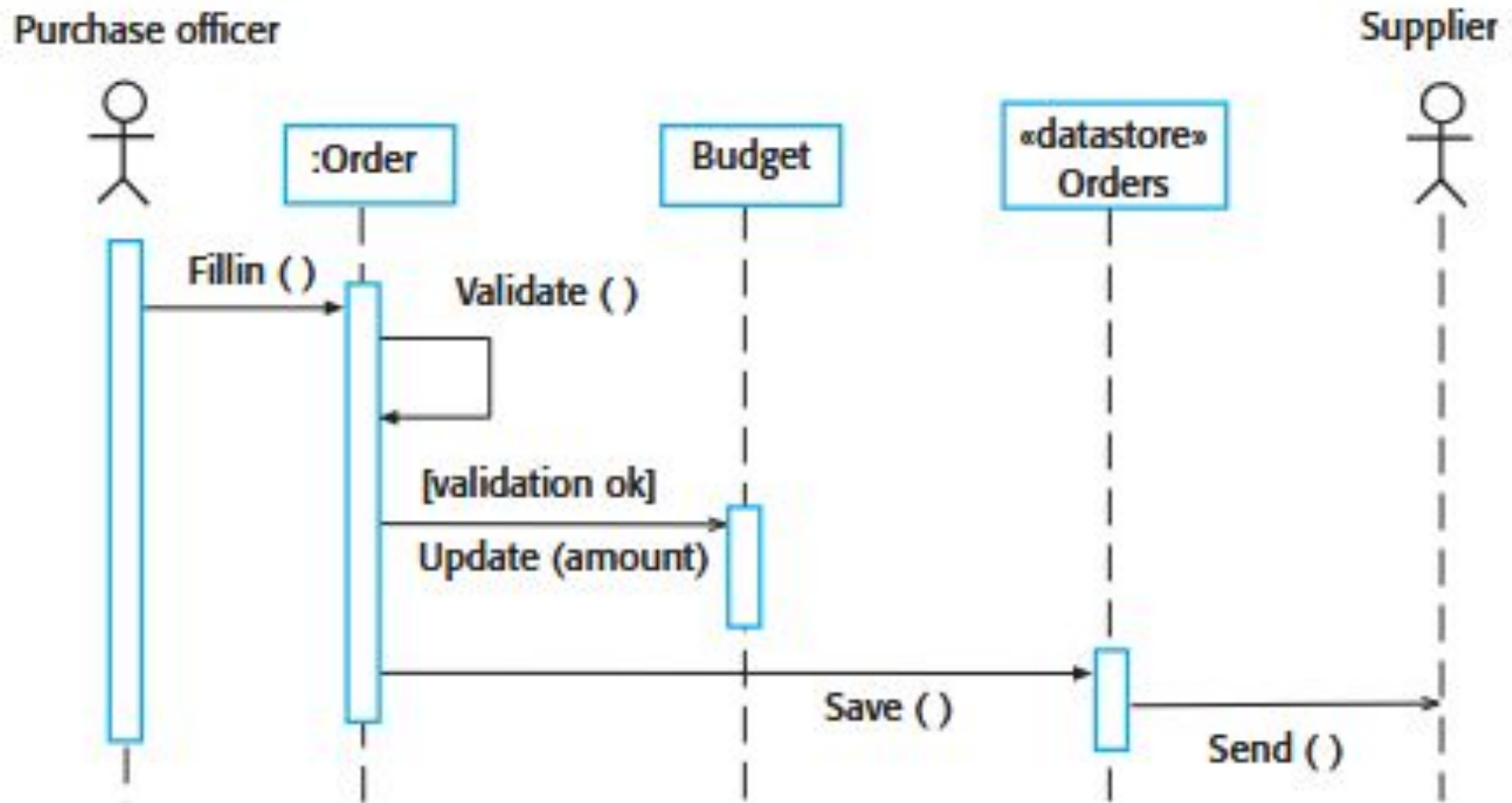
An activity model of the insulin pump's operation

Figure 5.15 illustrates this, using a sequence model of processing an order and sending it to a supplier.

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 oñ 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.

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 oñen 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.

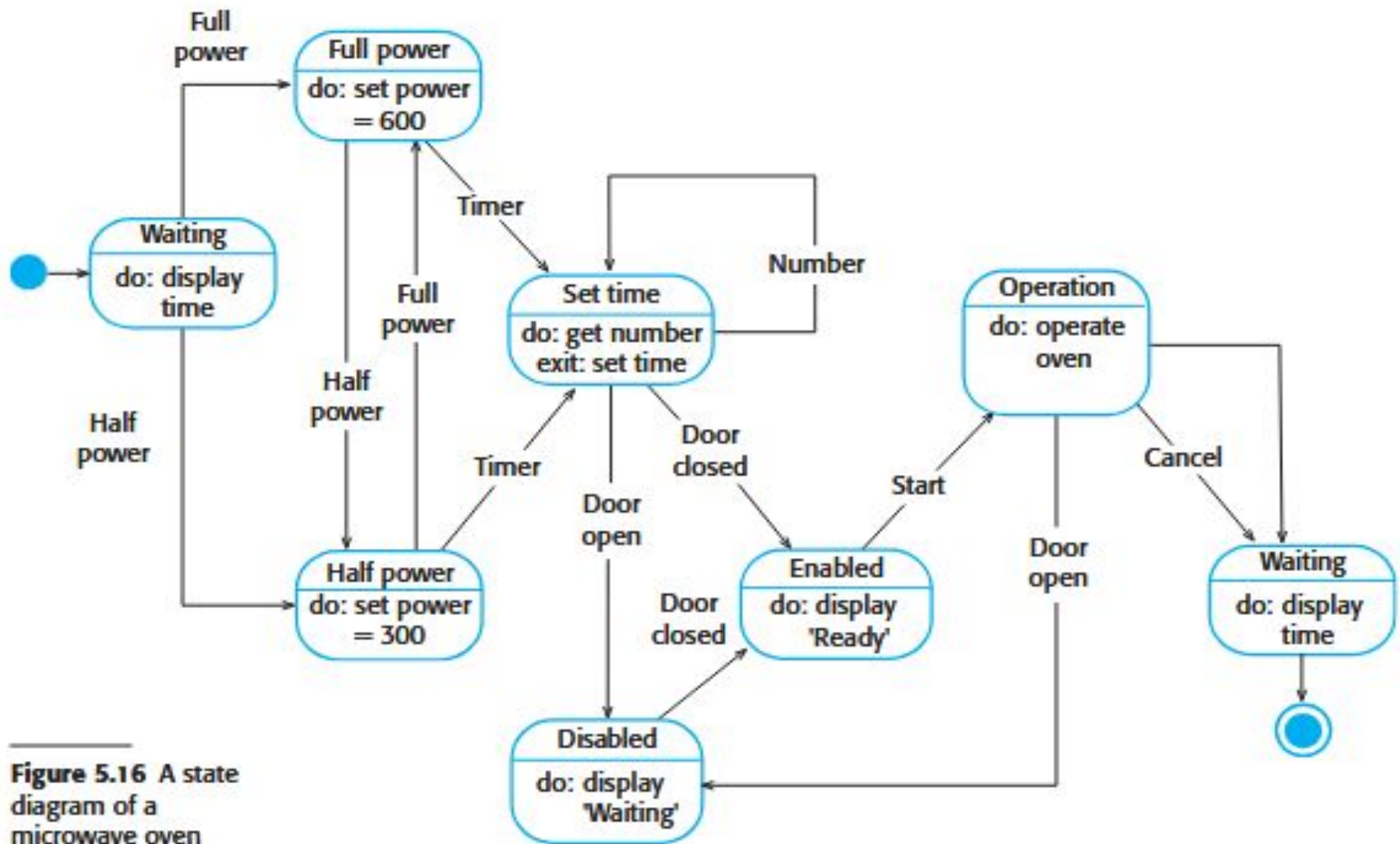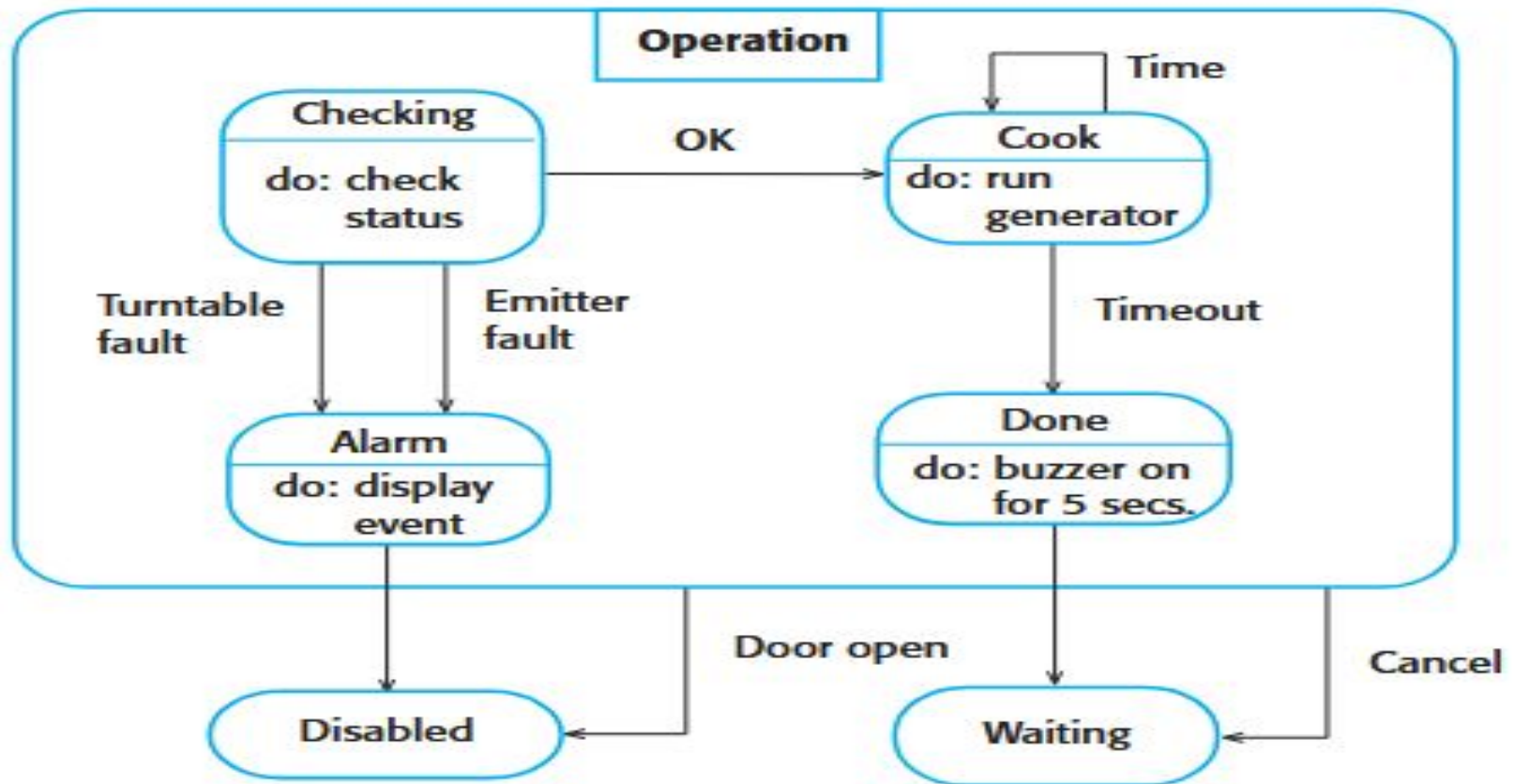❑ Statecharts are an integral part of the UML and are used to represent state machine models.

Figure 5.16 A state diagram of a microwave oven

A state model of the Operation 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.

Figure 5.18 States and stimuli for the microwave oven

Model-driven engineering

❑ Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.

❑ The programs that execute on a hardware/software platform are then generated automatically from the models.

❑ Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

Usage of model-driven **engineering**

❑ Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.

Pros

- Allows systems to be considered at higher levels of abstraction
- Generating code automatically means that it is cheaper to adapt systems to new platforms.

Cons

- Savings from generating code may be outweighed by the costs of developing translators for new platforms.
- Models for abstraction and not necessarily right for implementation.

Model driven architecture

❑ Model-driven architecture (MDA) was the precursor of more general model-driven engineering

❑ MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.

❑ Models at diñerent levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

Types of model

❖ Computation independent model (CIM)

These model the important domain abstractions used in a system. CIMs are sometimes called domain models.

❖ Platform independent model (PIM)

These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.

❖ Platform specific models (PSM)

These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.
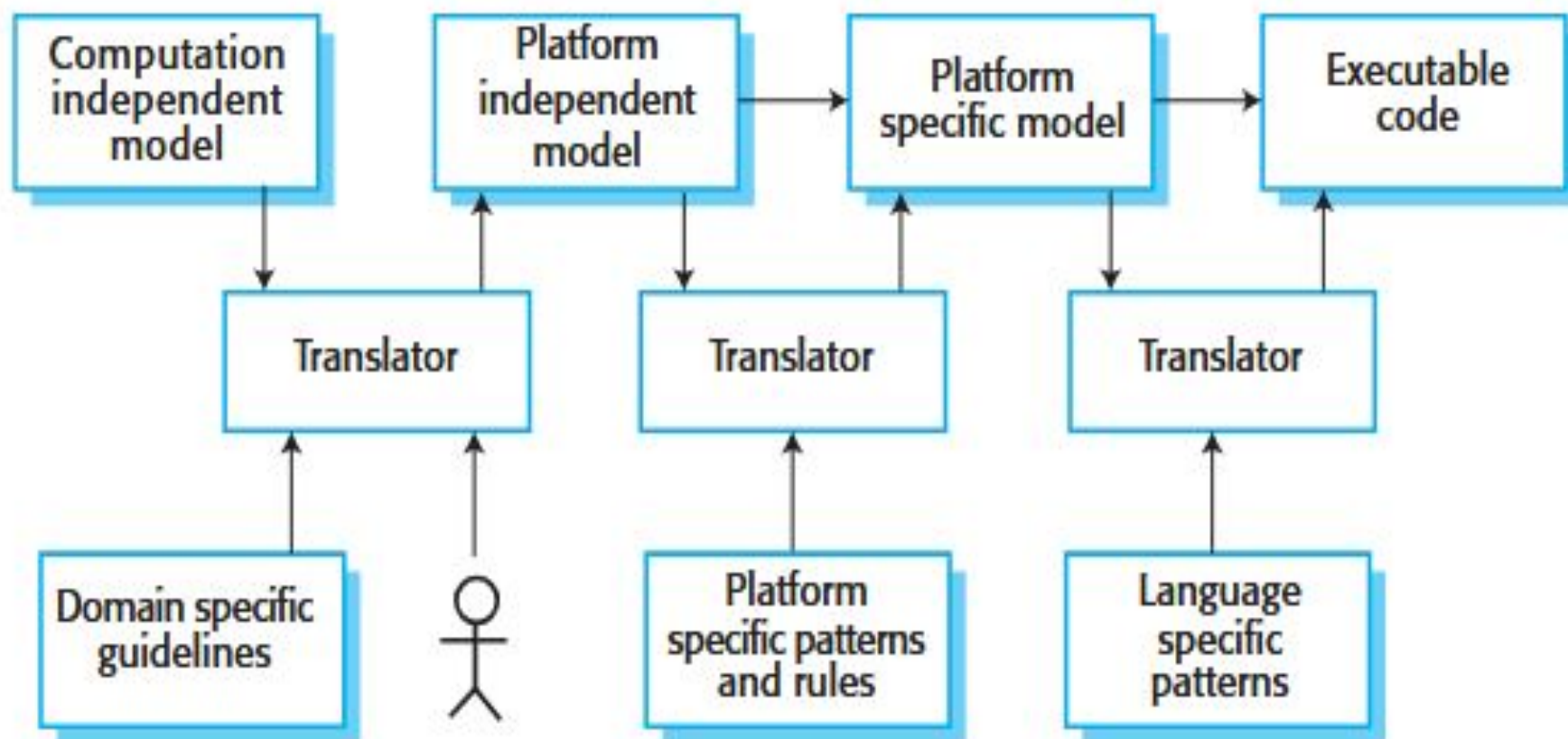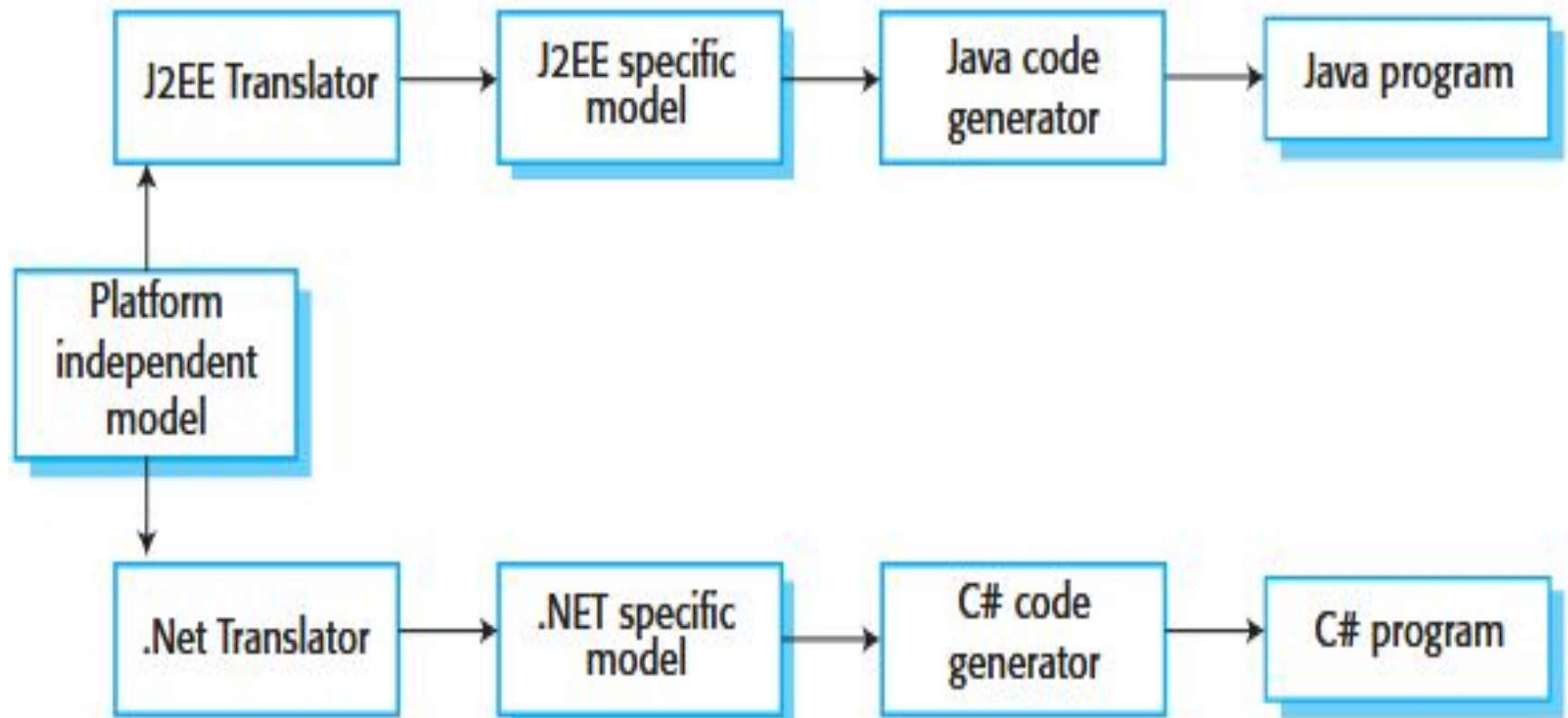
Figure 5.19 MDA transformations

Figure 5.20 Multiple platform-specific models

Agile methods and MDA

❑ The developers of MDA claim that it is intended to support an iterative approach to development and so can be used within agile methods.

❑ The notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto and I suspect that few agile developers feel comfortable with model- driven engineering.

❑ If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDA could be used in an agile development process as no separate coding would be required.

Adoption of MDA

✧ A range of factors has limited the adoption of MDE/MDA

✧ Specialized tool support is required to convert models from one level to another

✧ There is limited tool availability and organizations may require tool adaptation and customisation to their environment

✧ For the long-lifetime systems developed using MDA, companies are reluctant to develop their own tools or rely on small companies that may go out of business

Adoption of MDA

✧ Models are a good way of facilitating discussions about a software design. Howeverthe abstractions that are useful for discussions may not be the right abstractions for implementation.

✧ For most complex systems, implementation is not the major problem – requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

Adoption of MDA

✧ The arguments for platform-independence are only valid for large, long-lifetime systems. For software products and information systems, the savings from the use of MDA are likely to be outweighed by the costs of its introduction and tooling.

✧ The widespread adoption of agile methods over the same period that MDA was evolving has diverted attention away from model-driven approaches.

THANK YOU