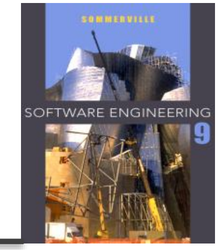
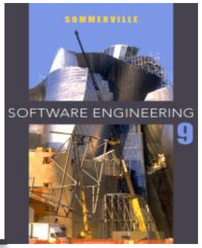


Chapter 5 – System Modeling

Topics covered



✧ Context models

✧ Interaction models

✧ Structural models

✧ Behavioral models

✧ Model-driven engineering

System modeling

✧ System modeling

- Developing abstract models of a system
- Each model presenting a different view or perspective of that system.
- Almost always based on notations in the Unified Modeling Language (UML).

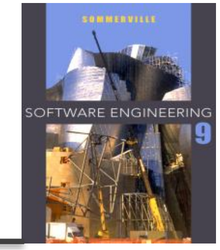
✧ Usage

- Helps analysts to understand the functionality of the system
- Are used to communicate with customers

Existing and planned system models

- ✧ Models of the existing system
  - Used during requirements engineering
  - They help clarify what the existing system does
  - Can be used as a basis for discussing its strengths and weaknesses
  - May lead to requirements for a new system
- ✧ Models of the new system
  - Used during requirements engineering to help explain the proposed requirements to other system stakeholders
  - Engineers use these models to discuss design proposals
  - 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

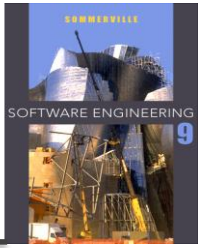


- An external perspective
  - where you model the context or environment of the system.
- An interaction perspective
  - where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective
  - where you model the organization of a system
  - 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
  - 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.

UML diagram types



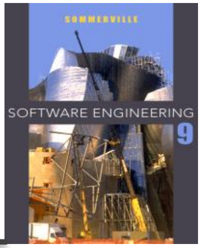
✧ Class diagrams

- show the object classes in the system and the associations between these classes.

✧ State diagrams

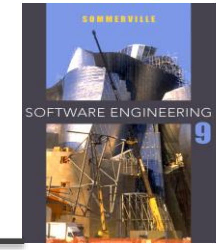
- show how the system reacts to internal and external events

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 have to be both correct and complete.

Context models

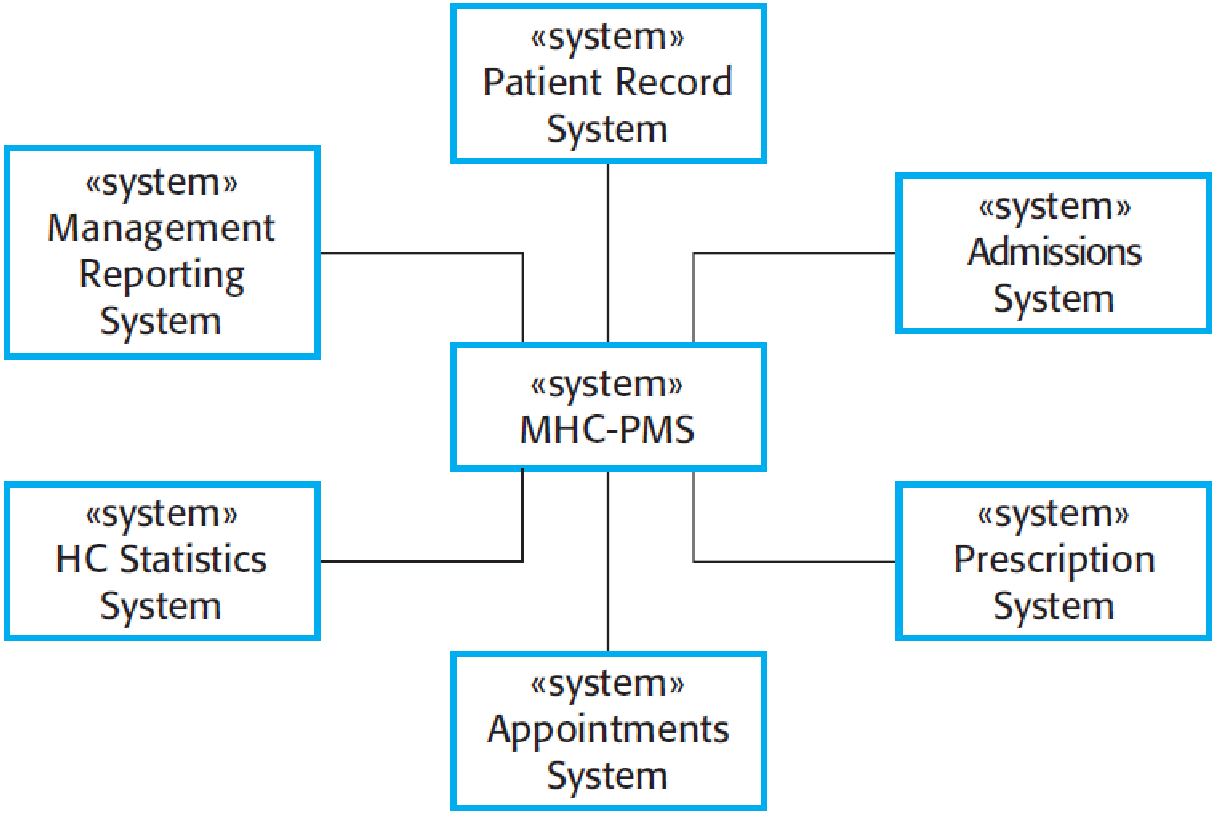


- ✧ Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- ✧ Social and organisational concerns may affect the decision on where to position system boundaries.
- ✧ Architectural models show the system and its relationship with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

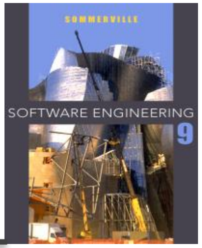
System boundaries

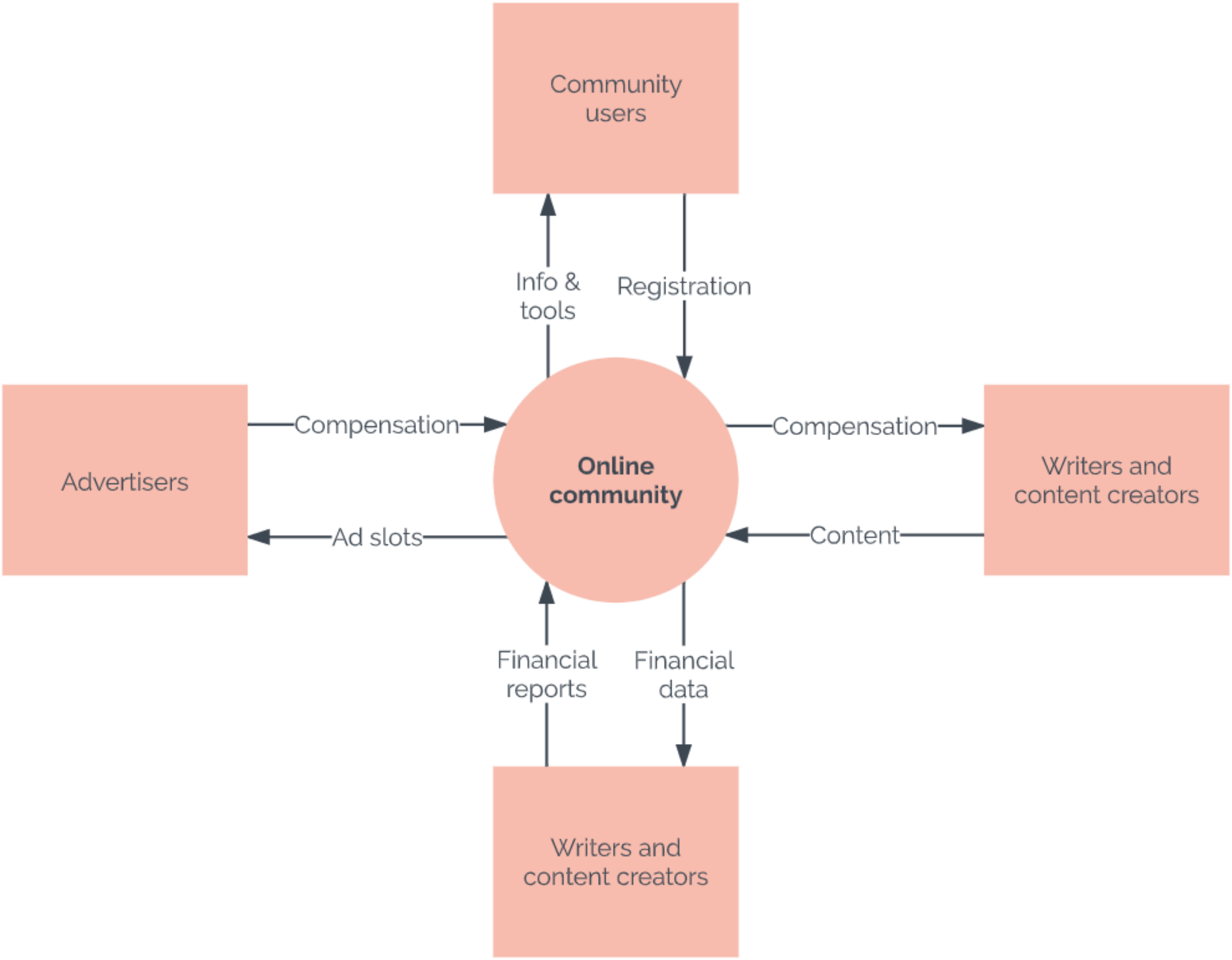
- ✧ System boundaries are established to define what is inside and what is outside the system.
  - They show other systems that are used or depend on the system being developed.
- ✧ The position of the system boundary has a profound effect on the system requirements.
- ✧ Defining a system boundary is 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 MHC-PMS



Data Flow Diagram

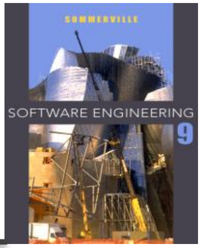


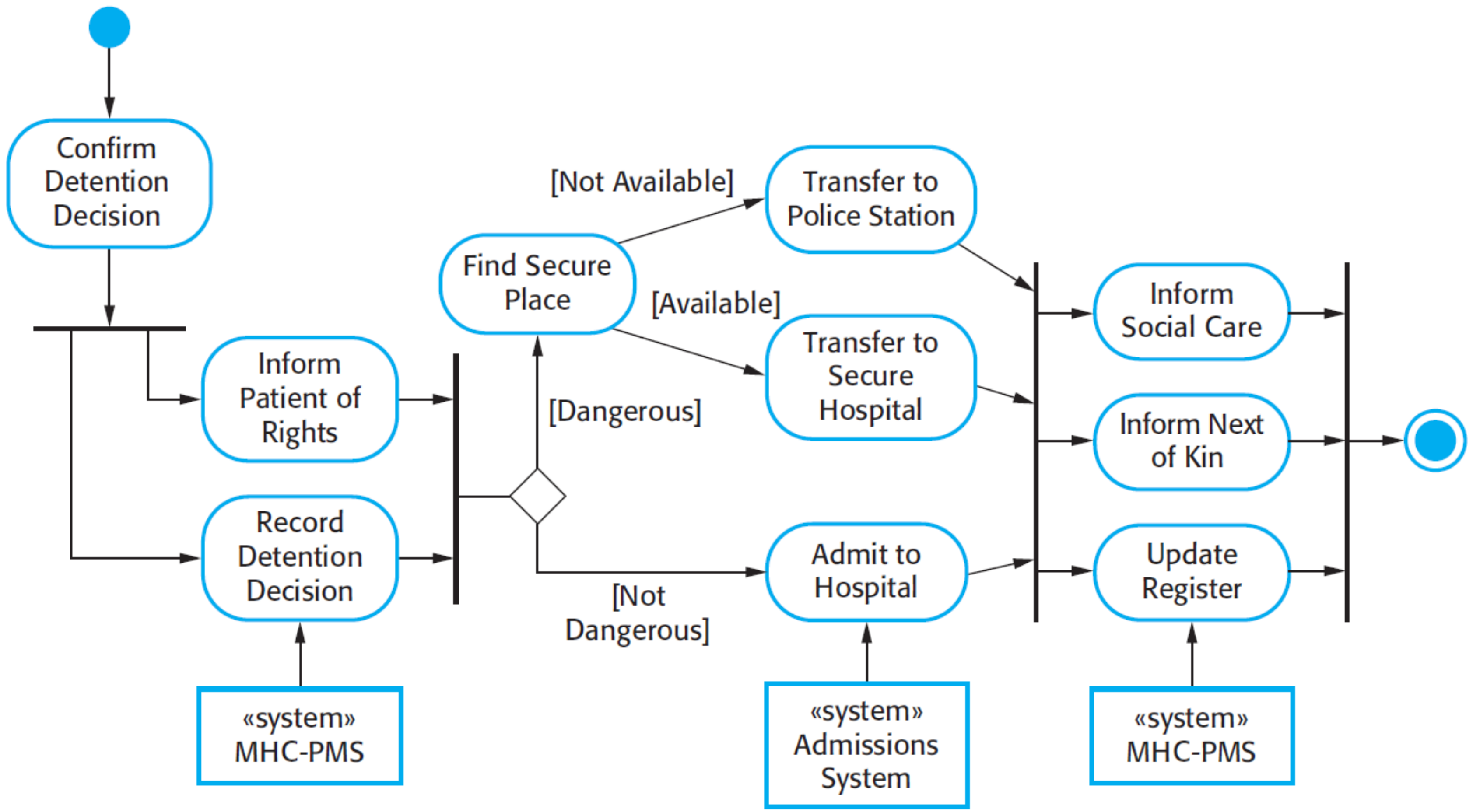


Process perspective

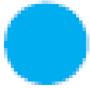
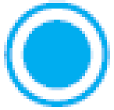
- Context models simply show the other systems in the environment
  - Not how the system being developed is used in that environment.
- Process models reveal how the system being developed is used in broader business processes.
- UML activity diagrams may be used to define business process models.

Process model of involuntary detention

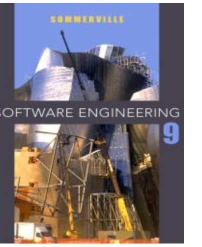




Activity diagram

- Activities that make up a system process
- Starts with 
- Ends with 
- Rounded rectangles show activities
- Arrows represent the flow of work
- Solid bar is used to indicate activity coordination (synchronization)

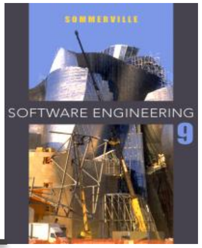
Activity diagram

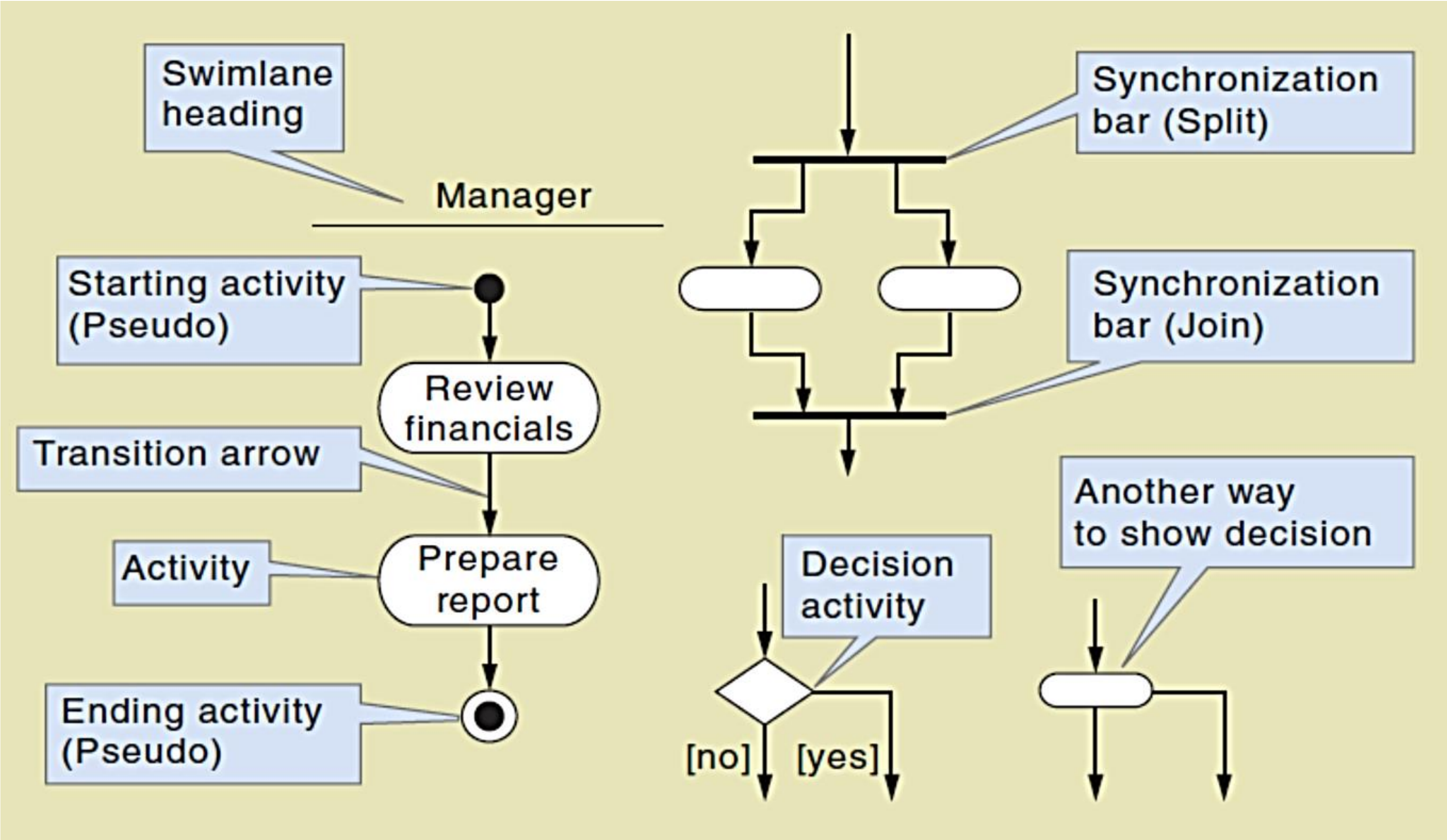


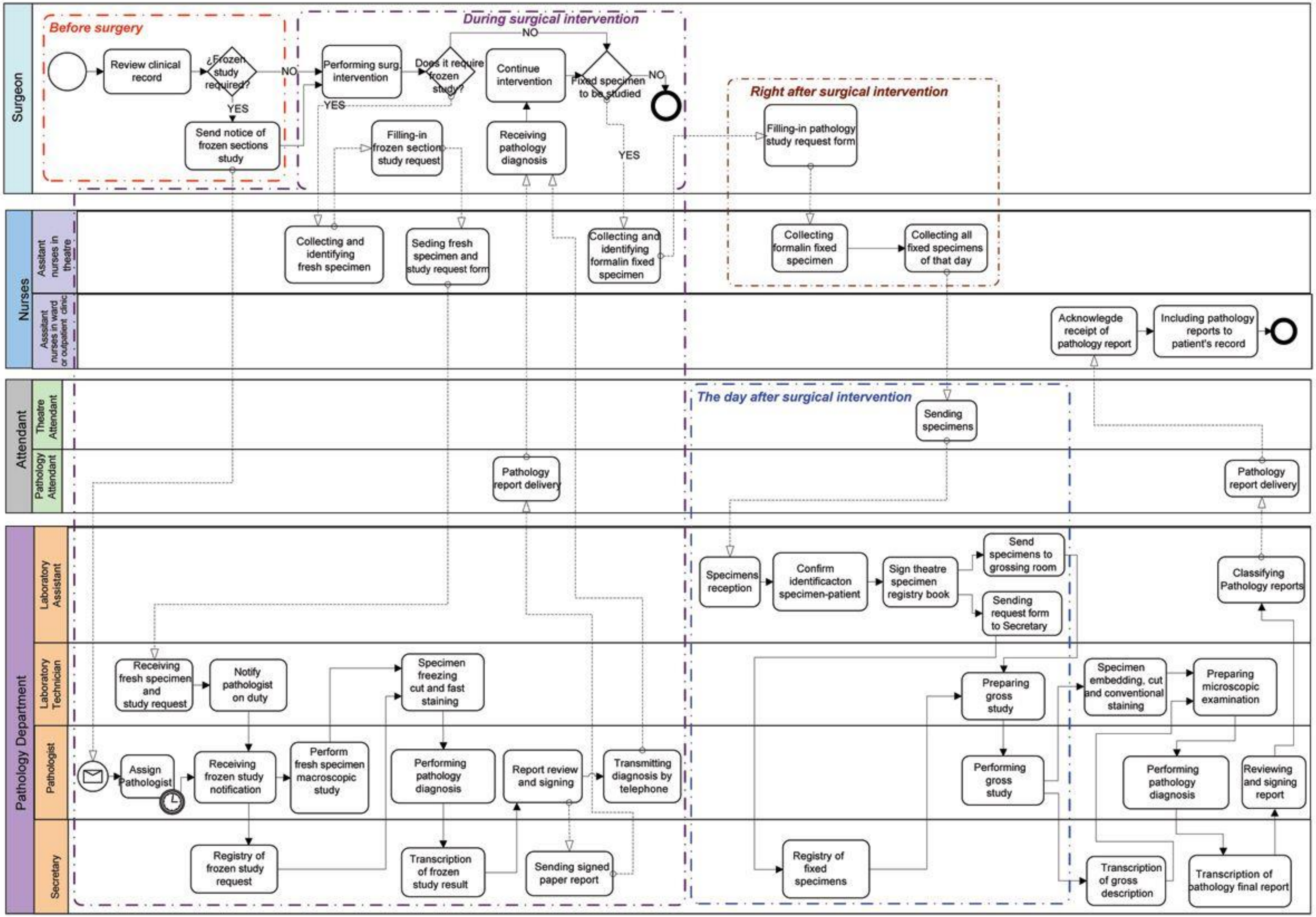
- Draw the activity diagram of coming to school in the morning

- From wake up time until you arrive to the school
- Consider parallel activities, conditions

Activity Diagram Symbols

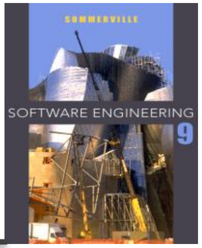






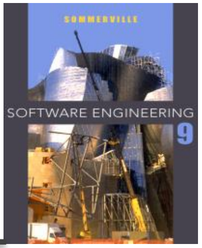
Surgeon
Before surgery
Review clinical record
¿Frozen study required?
NO
YES
Send notice of frozen sections study
During surgical intervention
Performing surg. intervention
Does it require frozen study?
YES
NO
Filling-in frozen section study request
Continue intervention
Receiving pathology diagnosis
Fixed specimen to be studied
NO
YES
Right after surgical intervention
Filling-in pathology study request form
Nurses
Assitant nurses in theatre
Collecting and identifying fresh specimen
Seding fresh specimen and study request form
Collecting and identifying formalin fixed specimen
Collecting formalin fixed specimen
Collecting all fixed specimens of that day
Asssitant nurses in ward or outpatient clinic
Acknowlegde receipt of pathology report
Including pathology reports to patient's record
Attendant
Theatre Attendant
The day after surgical intervention
Sending specimens
Pathology Attendant
Pathology report delivery
Pathology report delivery
Pathology Department
Laboratory Assistant
Specimens reception
Confirm identificacton specimen-patient
Sign theatre specimen registry book
Send specimens to grossing room
Sending request form to Secretary
Classifying Pathology reports
Laboratory Technician
Receiving fresh specimen and study request
Notify pathologist on duty
Specimen freezing cut and fast staining
Preparing gross study
Specimen embedding, cut and conventional staining
Preparing microscopic examination
Pathologist
Assign Pathologist
Receiving frozen study notification
Perform fresh specimen macroscopic study
Performing pathology diagnosis
Report review and signing
Transmitting diagnosis by telephone
Performing gross study
Performing pathology diagnosis
Reviewing and signing report
Secretary
Registry of frozen study request
Transcription of frozen study result
Sending signed paper report
Registry of fixed specimens
Transcription of gross description
Transcription of pathology final report

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.

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.

Example

✧ Draw a use case diagram for a simple online banking system

✧ Include use cases such as:

- Withdraw
- Deposit
- Transfer
- Login

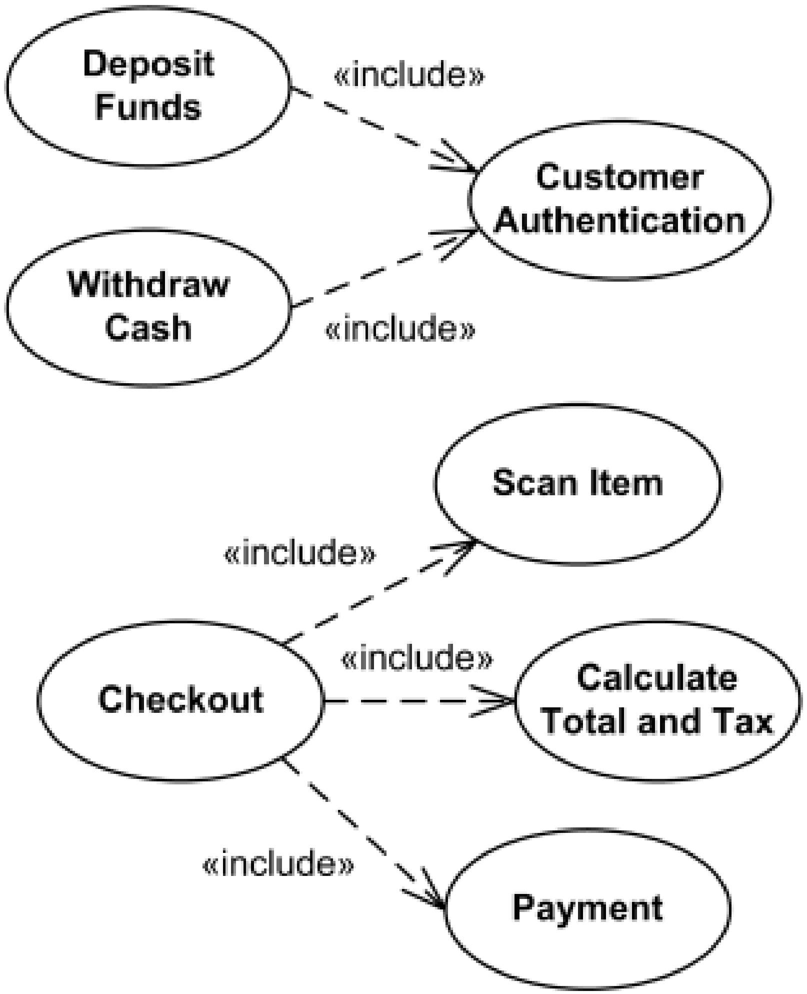
✧ Who are the actors?

- Include two actors

Include relationship

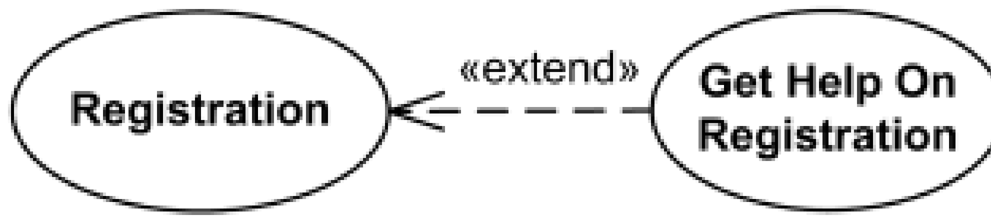
✧ Use:

- when there are **common parts** of the behavior of two or more use cases,
- to simplify large use case by splitting it into several use cases.



Extend relationship

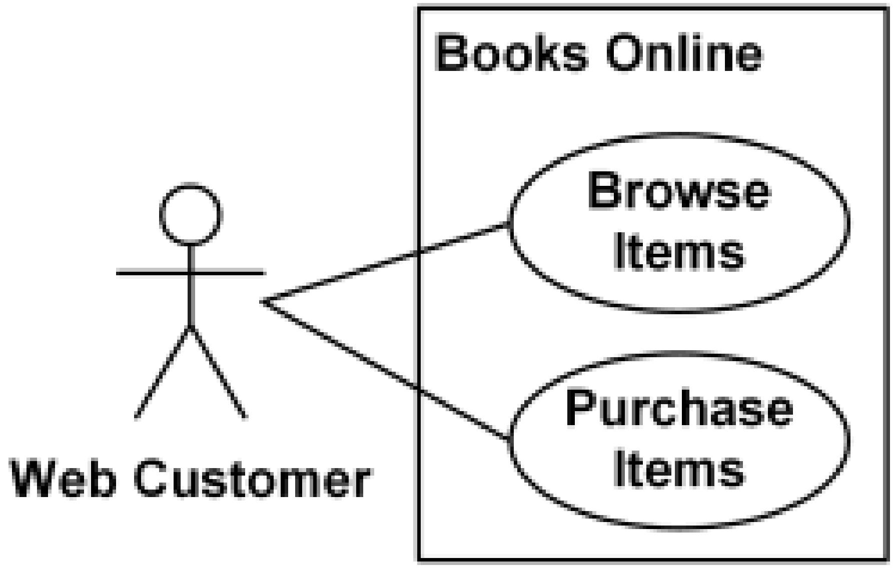
- **Extended** use case is meaningful on its own
- it is **independent** of the extending use case.
- **Extending** use case typicallydefines **optional** behavior that is not necessarily meaningful by itself.



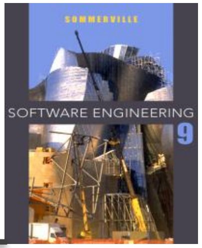
Subject – system boundary

- **Subject** is presented by a rectangle with subject name in upper corner

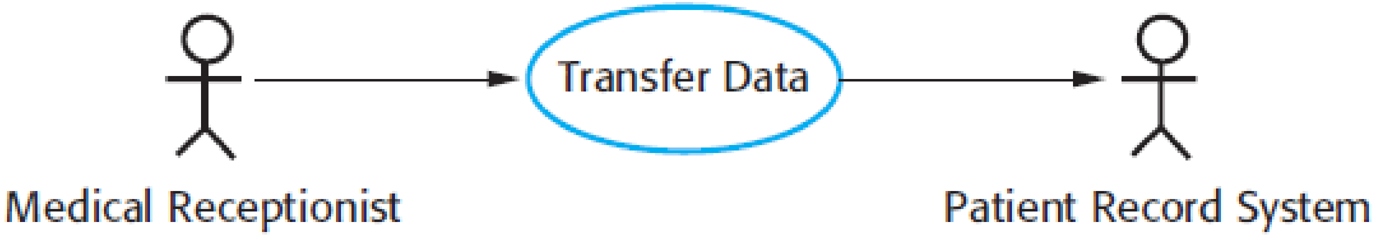
- **Use cases** inside the rectangle and **actors** - outside of the system boundaries.



Transfer-data use case



✧ A use case in the MHC-PMS

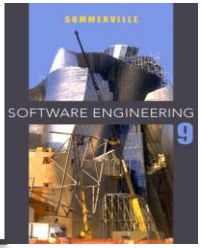


✧ High level overview

✧ Details can be provided as

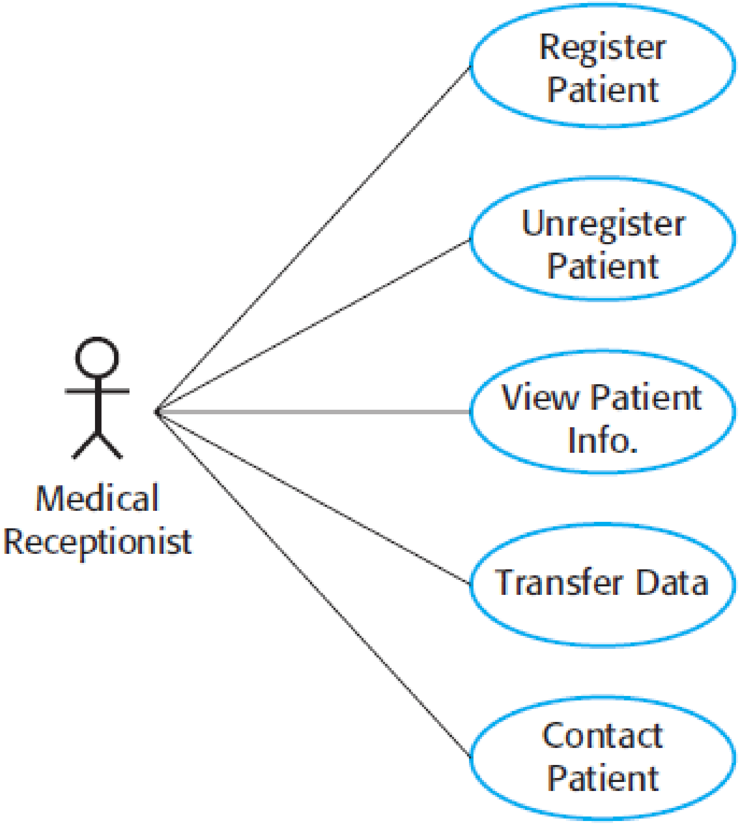
- Text
- Table
- Sequence diagram

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 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.

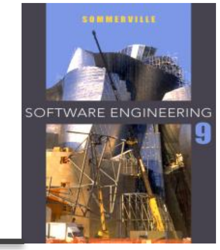
Use cases in the MHC-PMS involving the role 'Medical Receptionist'



Practice!

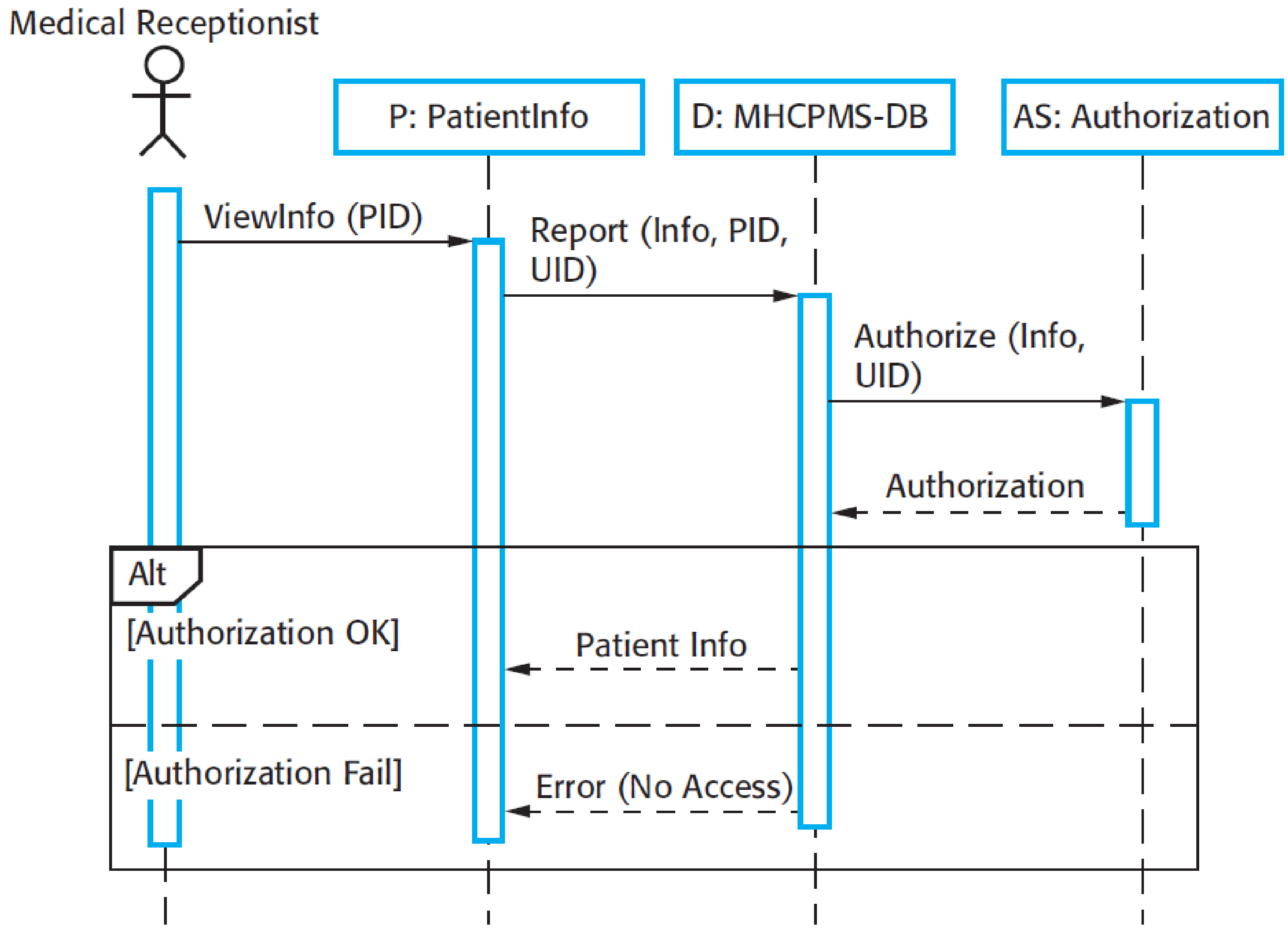
- Draw use case diagram for your system
- You can use the diagram from last week as basis

Sequence diagrams

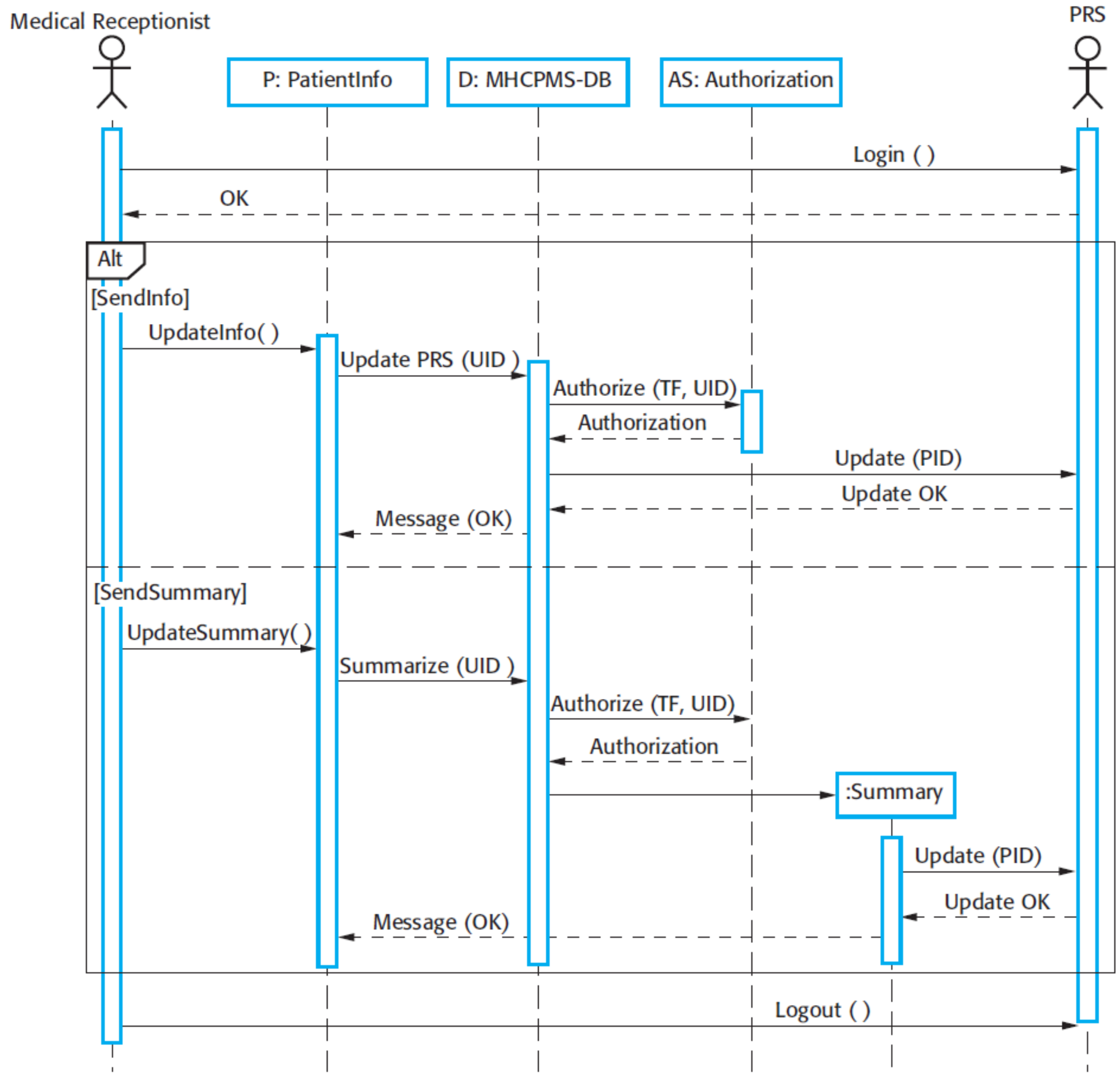


- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The objects and actors involved are listed along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

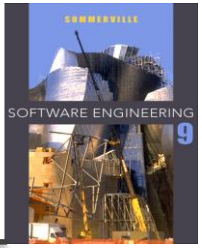
Sequence diagram for View patient information



Sequence diagram for Transfer Data

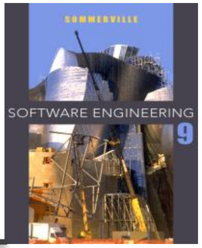


Structural models



- Structural models 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.
- Structural models are useful in discussing and designing the system architecture.

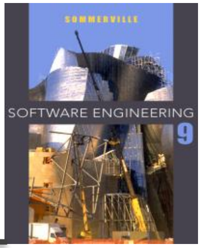
Class diagrams

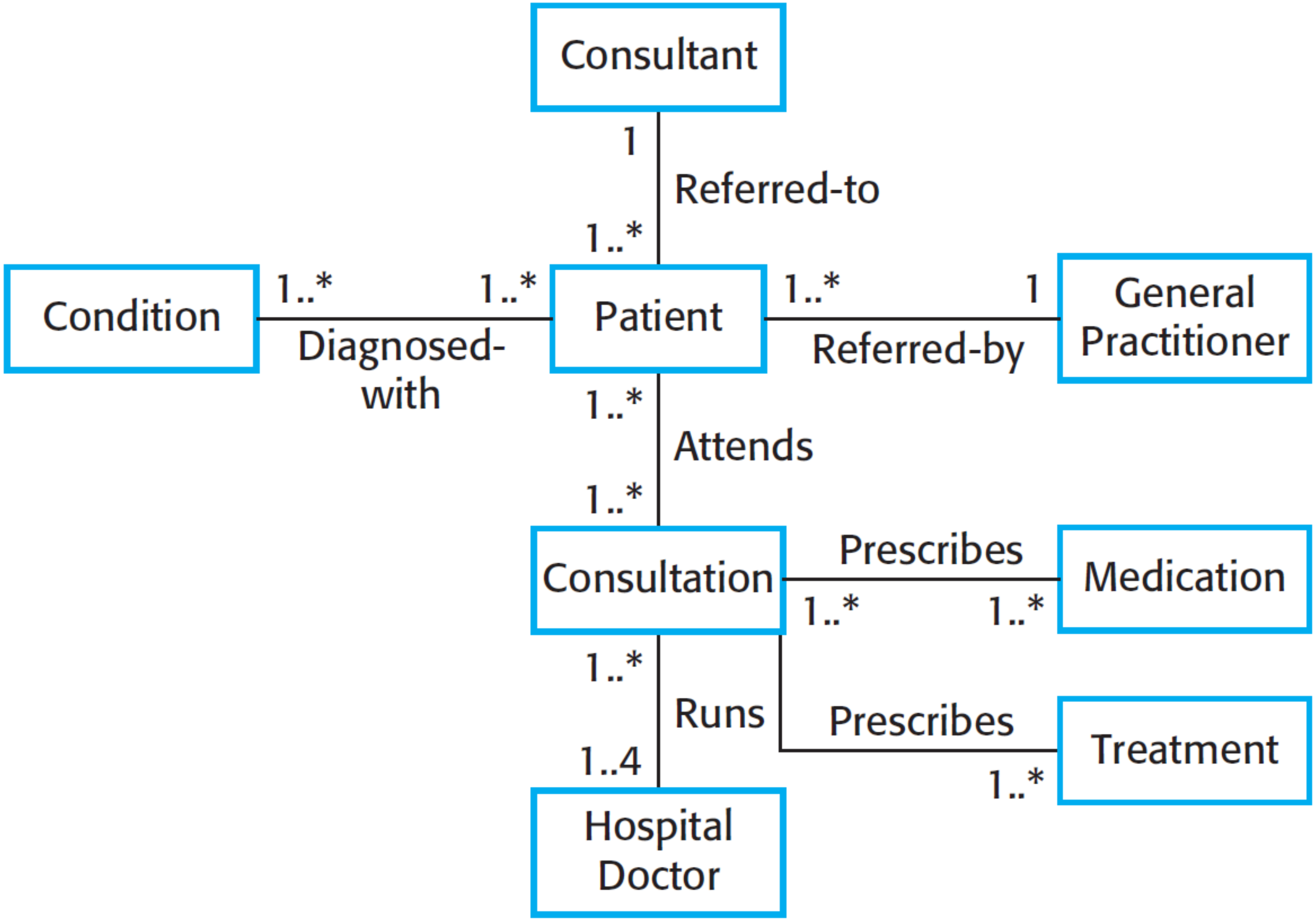


- ✧ Used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- ✧ An object class can be thought of as a general definition of one kind of system object.
- ✧ An association is a link between classes that indicates that there is some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

UML classes and association

Classes and associations in the MHC-PMS





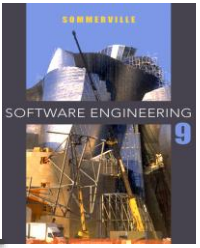
The Consultation class

Consultation

Doctors
Date
Time
Clinic
Reason
Medication Prescribed
Treatment Prescribed
Voice Notes
Transcript
...

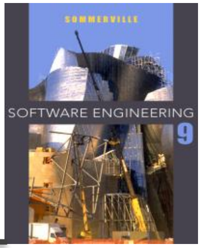
New ( )
Prescribe ( )
RecordNotes ( )
Transcribe ( )
...

Key points



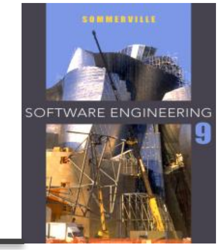
- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behaviour.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

Generalization



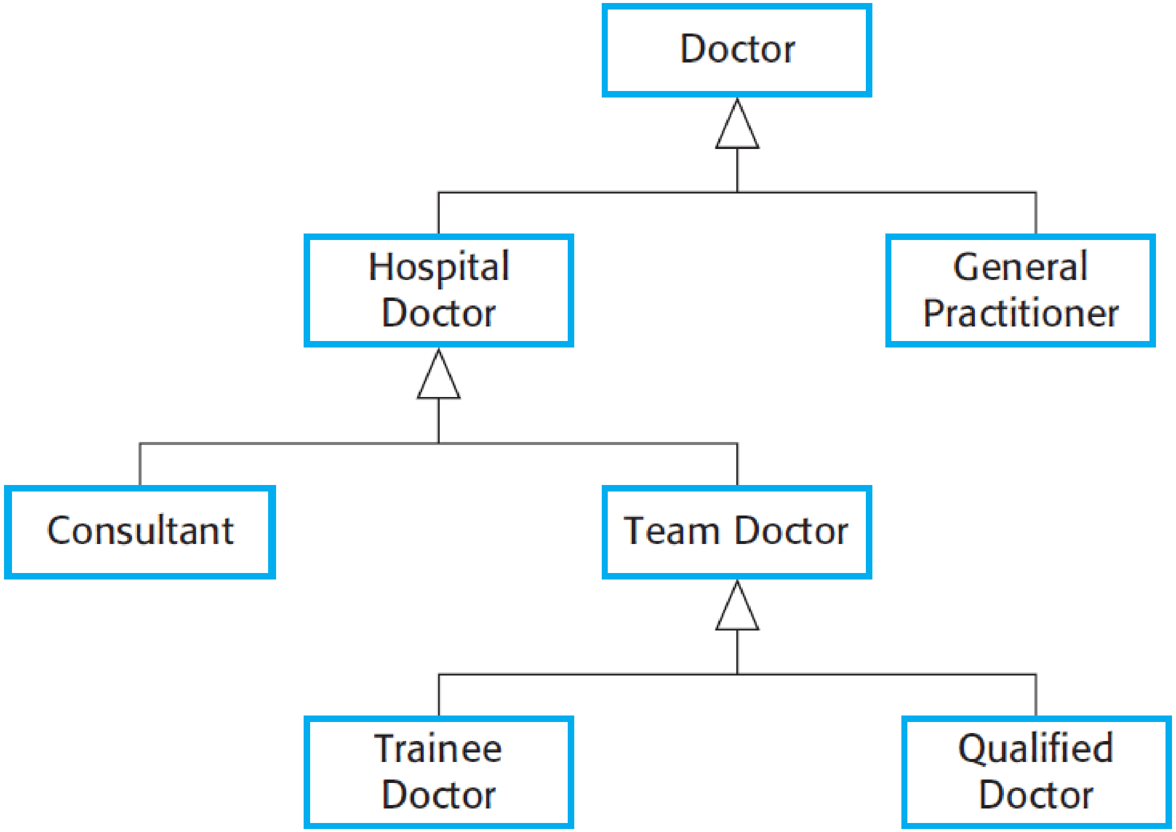
- Generalization is an everyday technique that we use to manage complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- This allows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

Generalization

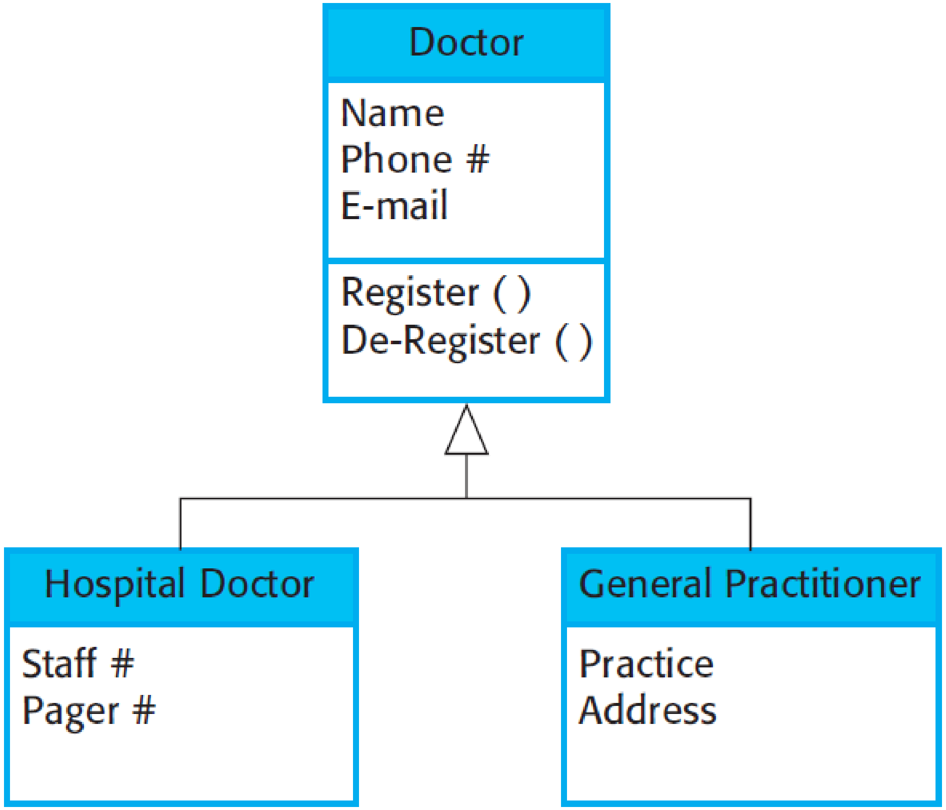


- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization.
  - If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In object-oriented languages, such as Java, generalization 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.

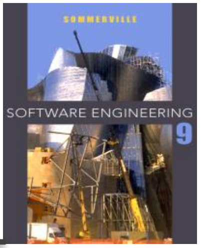
A generalization hierarchy



A generalization hierarchy with added detail



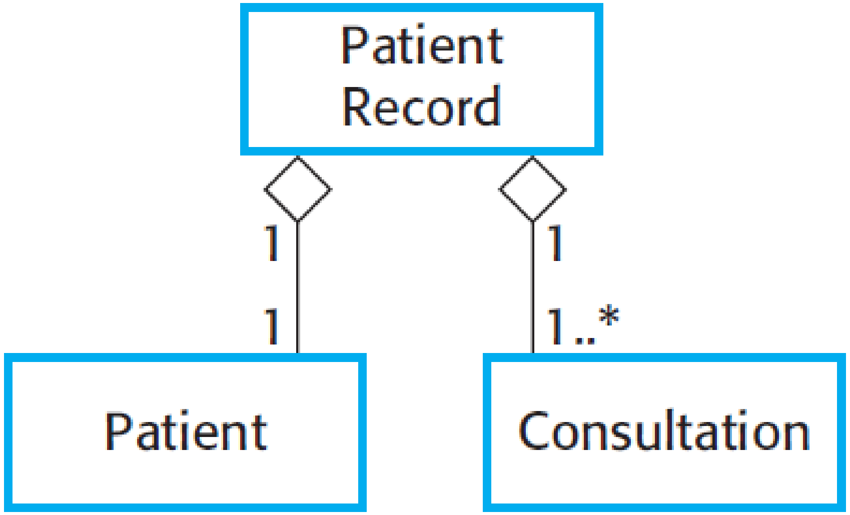
Object class aggregation models



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

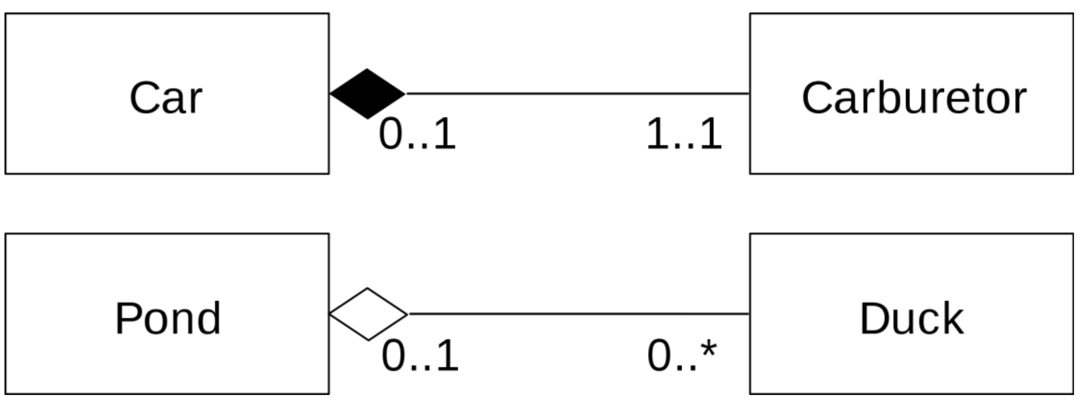
- Aggregation models are similar to the part-of relationship in semantic data models.

The aggregation association



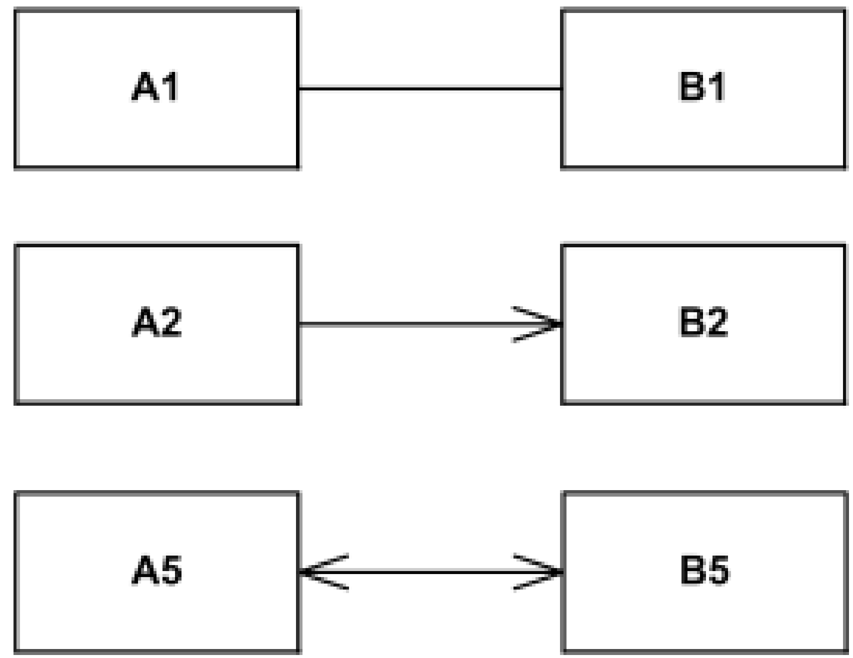
Aggregation vs Composition

- ✧ A university is a composition of classes
  - University ceases to exist -> classes ceases to exist
- ✧ A university is a aggregation of professors/students
  - University ceases to exist -> professors/students continue to exist



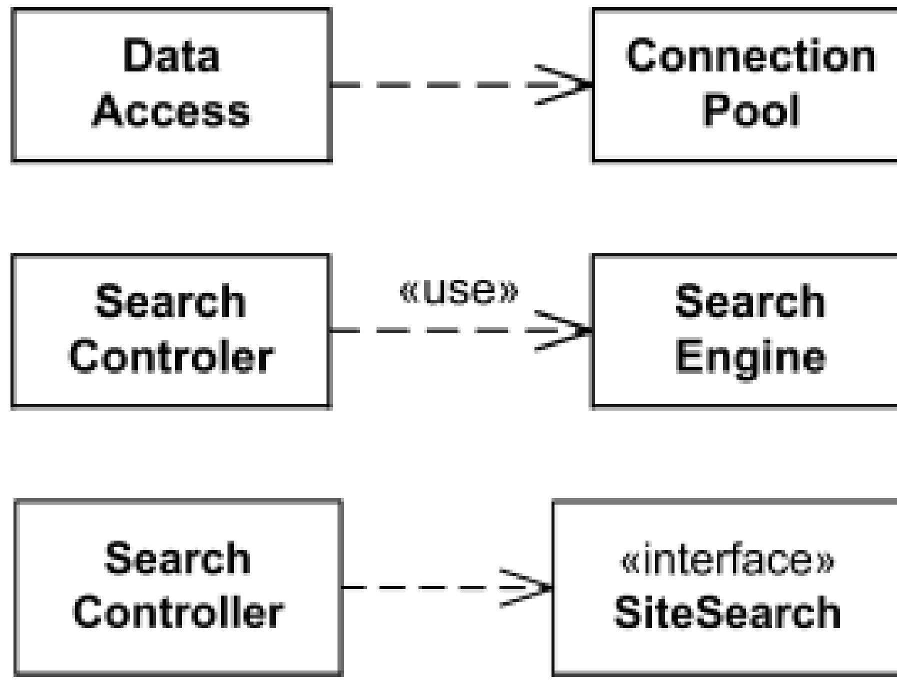
Association navigability

- End property of association is navigable from the opposite end(s) of association if instances of the classifier at this end of the link can be accessed efficiently at runtime from instances at the other ends of the link.

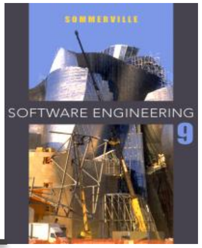


Dependency

✧ The model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier)

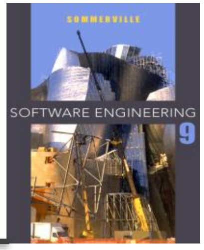


Behavioral models



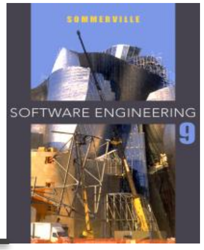
- ✧ Models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- ✧ You can think of these stimuli as being of two types:
  - Data Some data arrives that has to be processed by the system.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

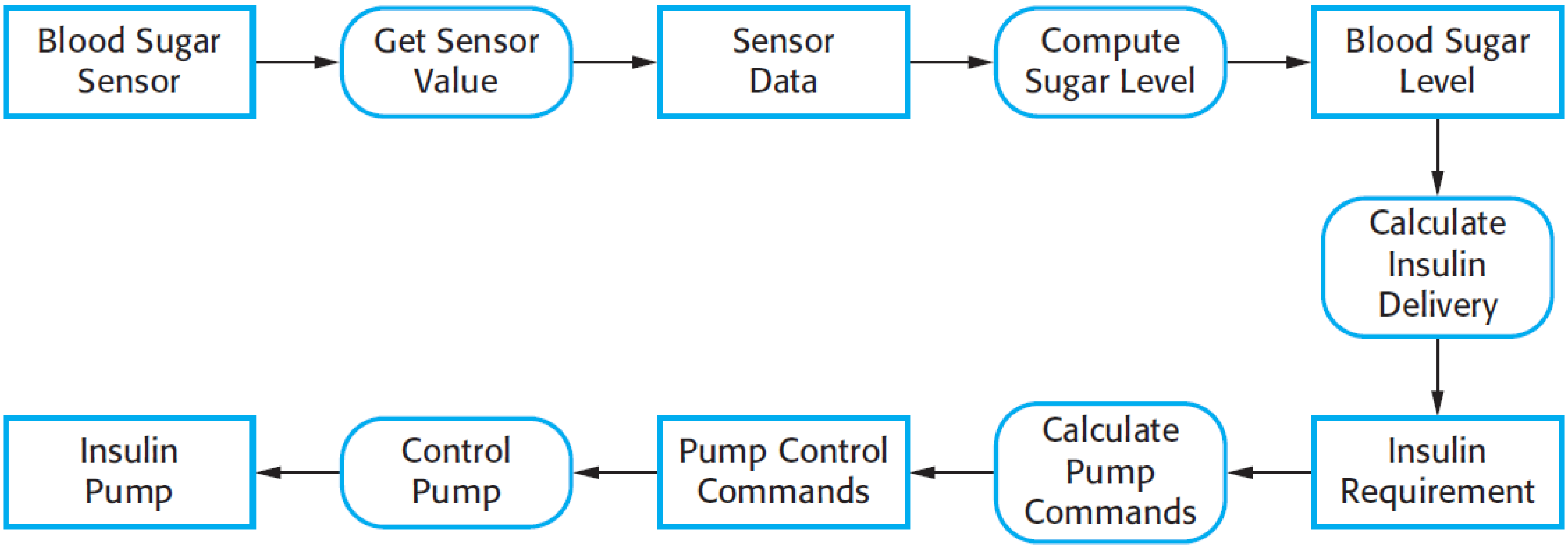
Data-driven modeling



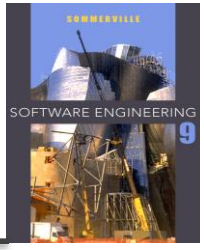
- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated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

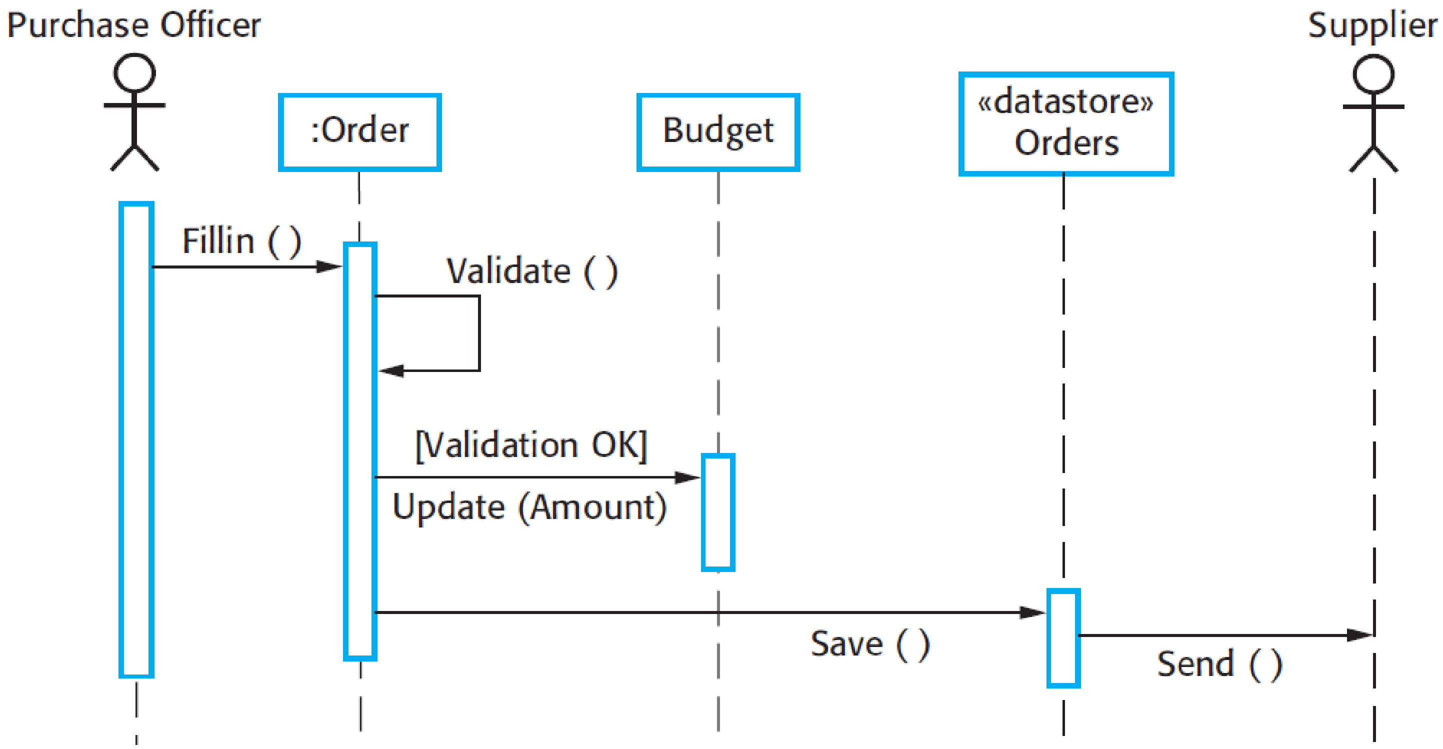
An activity model of the insulin pump's operation



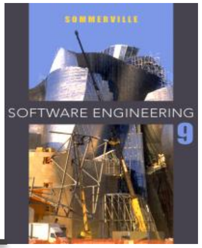


Order processing





Event-driven modeling

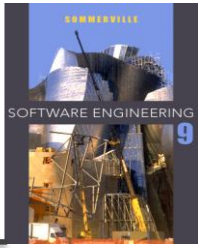


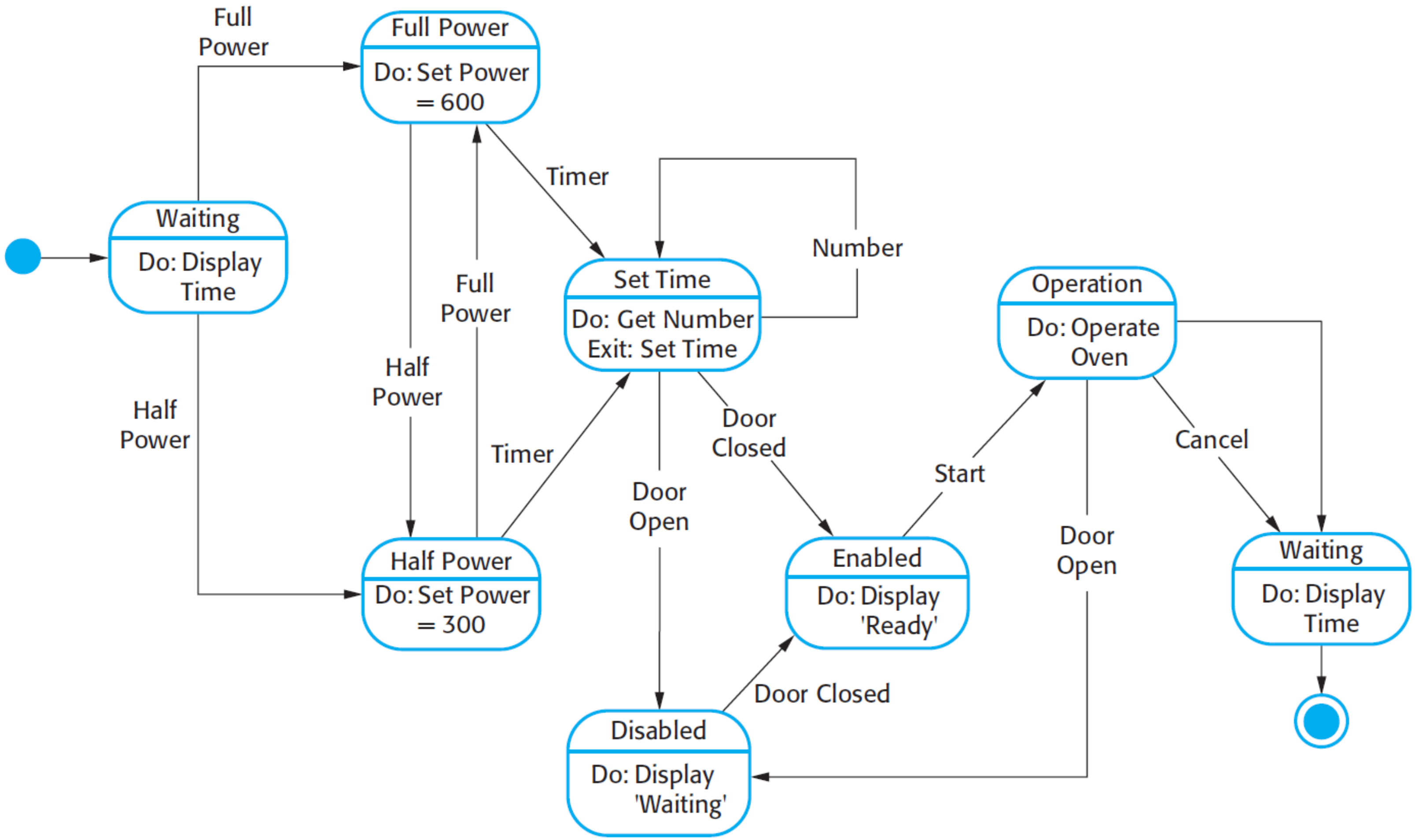
- Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as ‘receiver off hook’ by generating a dial tone.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the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

State machine models

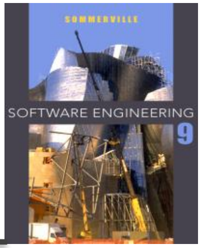
- These model the behaviour of the system in response to external and internal events.
- They show the system’s responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the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

State diagram of a microwave oven



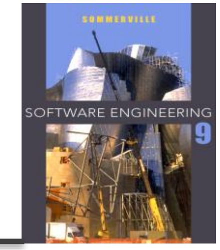


States and stimuli for the microwave oven (b)



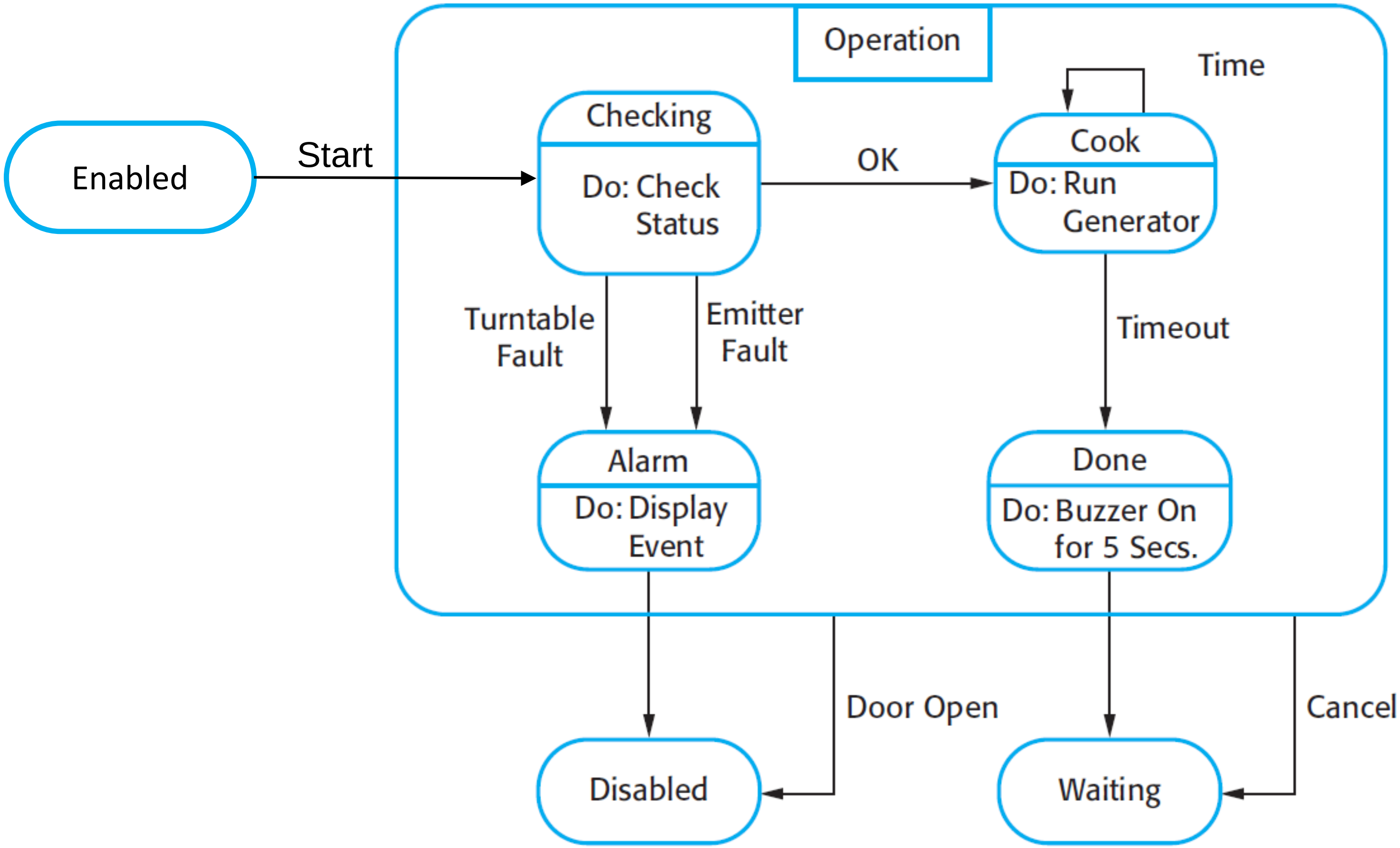
Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

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.

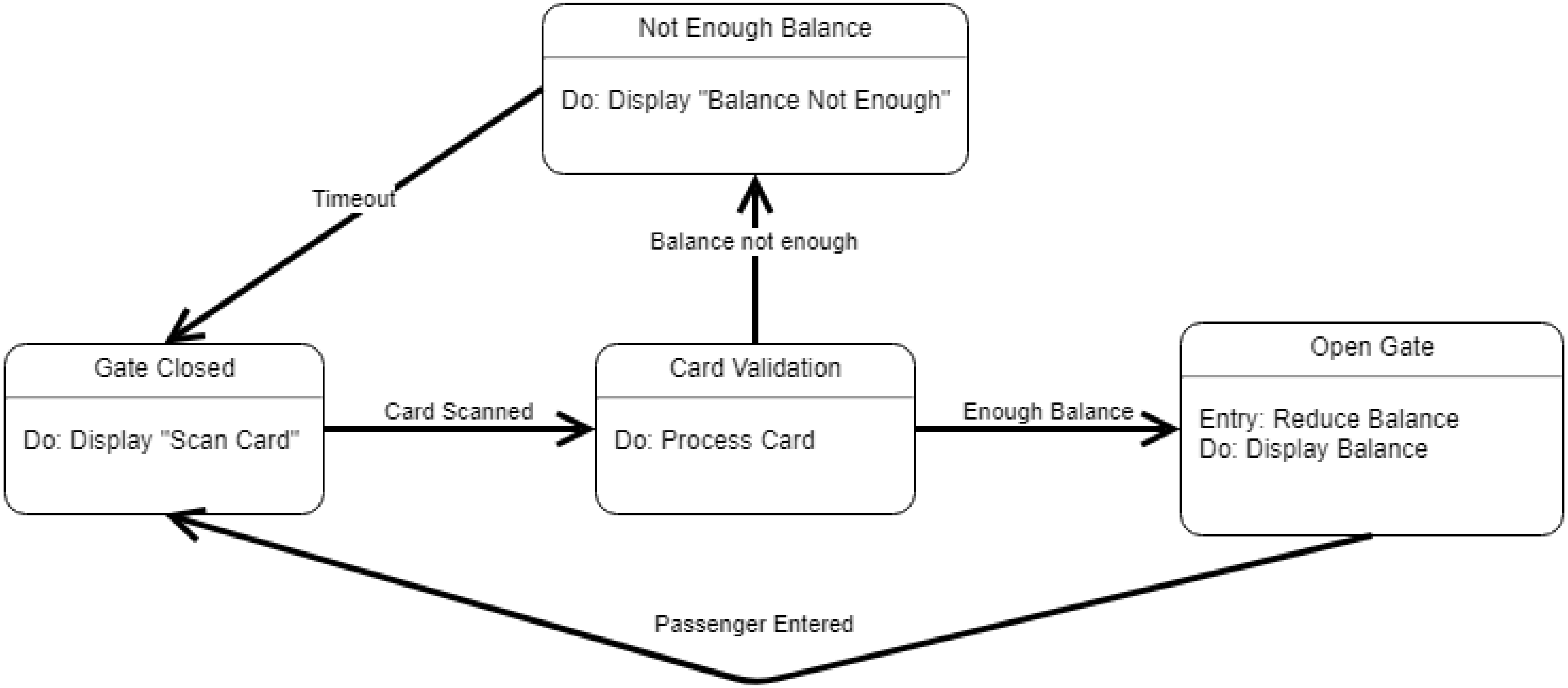
Microwave oven operation



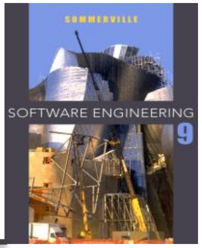
Metro pay gate (Quiz)

✧ Draw state machine diagram for metro gates in Tehran.

Metro pay gate

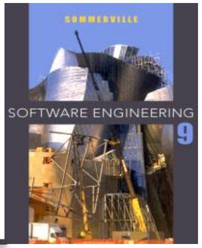


Model-driven engineering



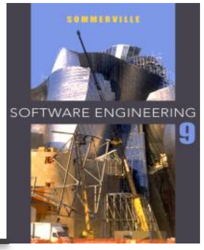
- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that 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.

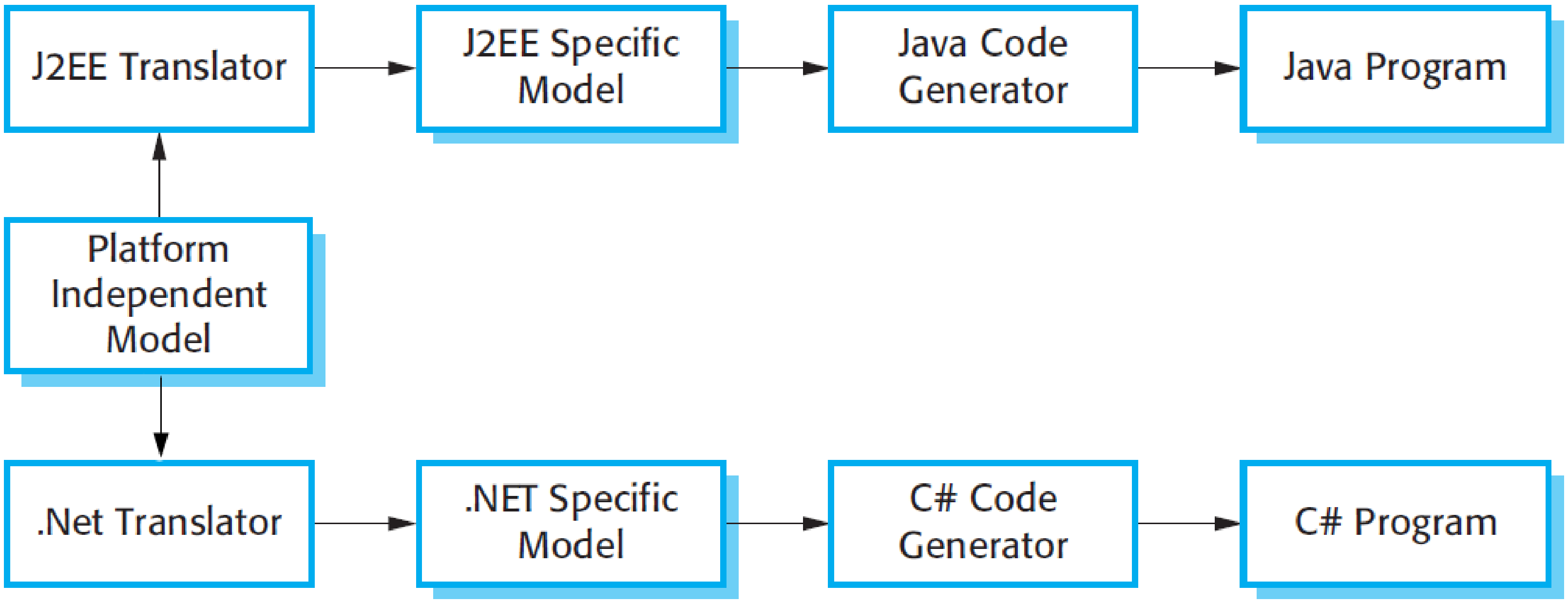
Usage of model-driven engineering



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

Multiple platform-specific models



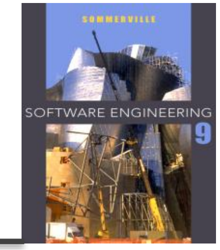


Executable UML

- The fundamental notion behind model-driven engineering is that completely automated transformation of models to code should be possible.

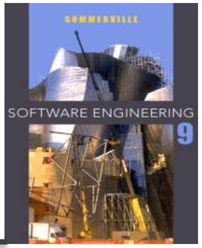
- This is possible using a subset of UML 2, called Executable UML or xUML.

Features of executable UML



- To create an executable subset of UML, the number of model types has therefore been dramatically reduced to these 3 key types:
  - Domain models that identify the principal concerns in a system. They are defined using UML class diagrams and include objects, attributes and associations.
  - Class models in which classes are defined, along with their attributes and operations.
  - State models in which a state diagram is associated with each class and is used to describe the life cycle of the class.

Key points



- Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system's behavior in response to internal or external events.
- Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.