Software Engineering

Dr. Fatma ElSayed

Computer Science Department
fatma.elsayed@fci.bu.edu.eg

Course Information

Contents

Course Information

Contents

Chapter 5: System Modelling

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 modelling has generally come to mean representing the system using some kind of **graphical notation**, which is now almost always based on notations in the **Unified Modelling Language (UML).**

When to use:

- During the requirements engineering process: Models are used to help derive the requirements for a system.
- During the design process: Models are used to describe the system to engineers implementing the system.
- After implementation: to document the system’s structure and operation.

UML

- The Unified modeling Language is a set of ***13 different diagram types*** that may be used to model software systems.

1. Class Diagram
2. Object Diagram
3. Component Diagram
4. Composite Structure Diagram
5. Deployment Diagram
6. Package Diagram

Structural Diagrams

1. Use Case Diagram
2. Sequence Diagram
3. Activity Diagram
4. State Machine Diagram
5. Communication Diagram
6. Interaction Overview Diagram
7. Timing Diagram

Behavioral Diagrams

UML

- The Unified modeling Language is a set of ***13 different diagram types*** that may be used to model software systems.

1. Class Diagram
2. Object Diagram
3. Component Diagram
4. Composite Structure Diagram
5. Deployment Diagram
6. Package Diagram

Structural Diagrams

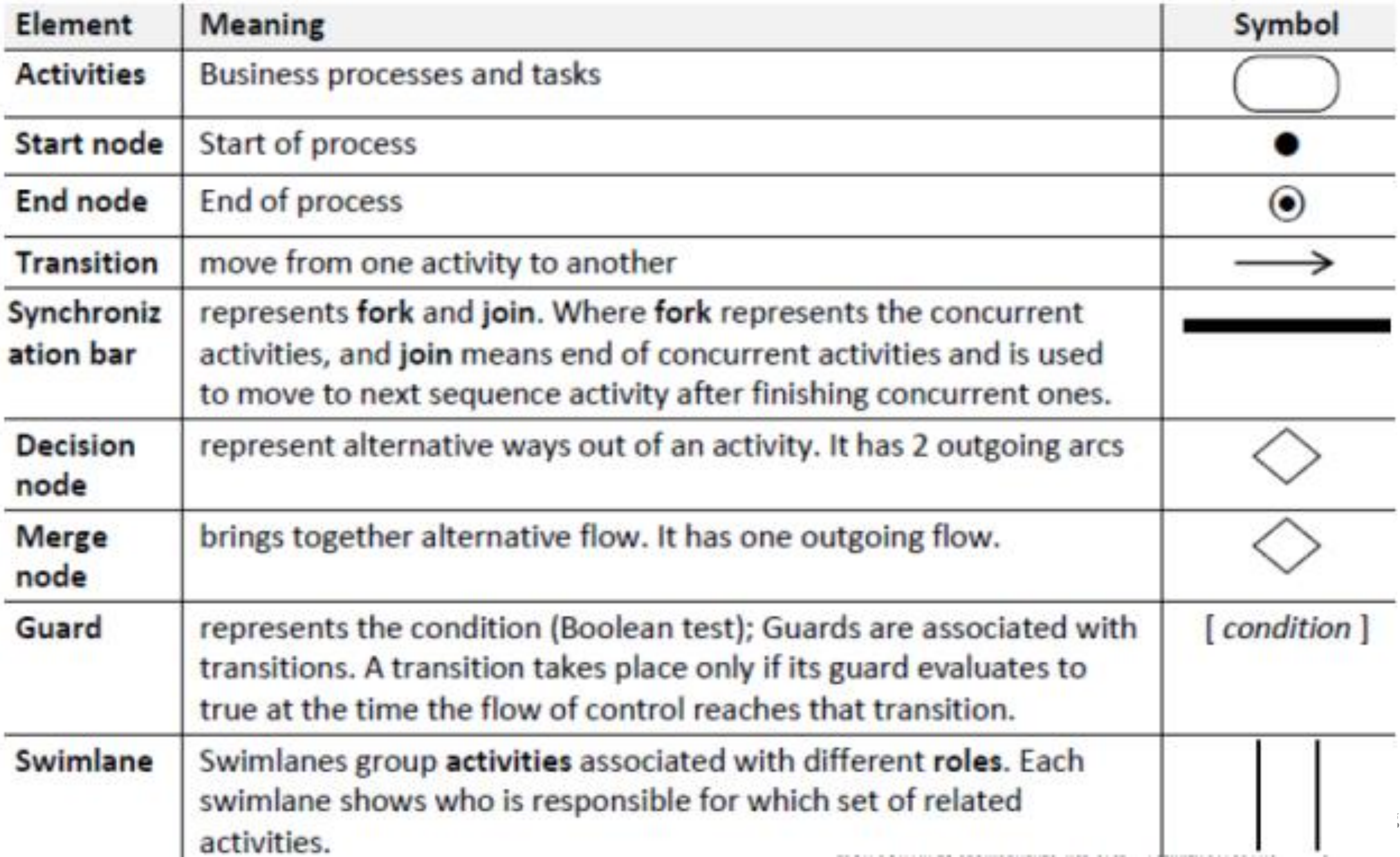
1. Use Case Diagram
2. Sequence Diagram
3. Activity Diagram
4. State Machine Diagram
5. Communication Diagram
6. Interaction Overview Diagram
7. Timing Diagram

Behavioral Diagrams

Activity Diagrams

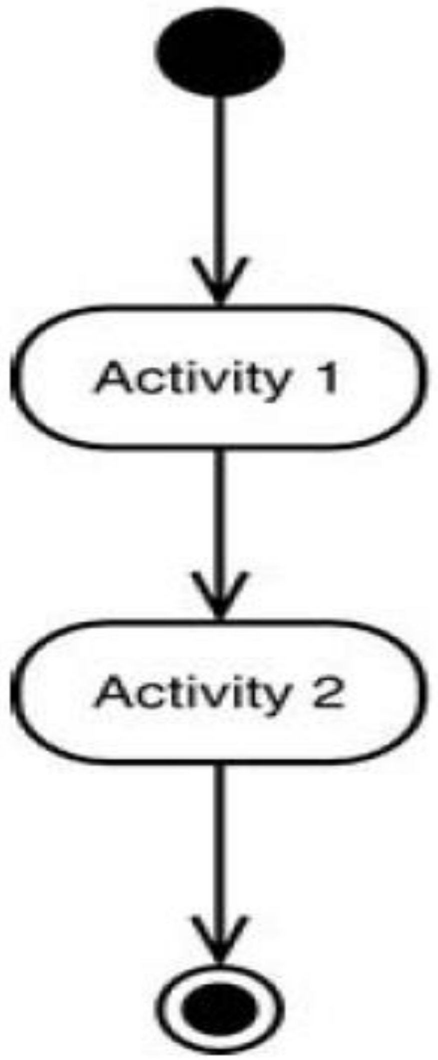
- Activity diagram is designed to be a simplified look at what happens during an operation or a process.
- This diagram much like the flowcharts . It shows steps **(called, activities),** showing their sequences, and where activities can be carried out in parallel. as well as decision points and branches.
- It can also be extended to show which **person** is responsible for which activity.
- An activity could be a task that a *person* **or** *a computer* might perform.

Activity Diagrams: Symbols and Elements

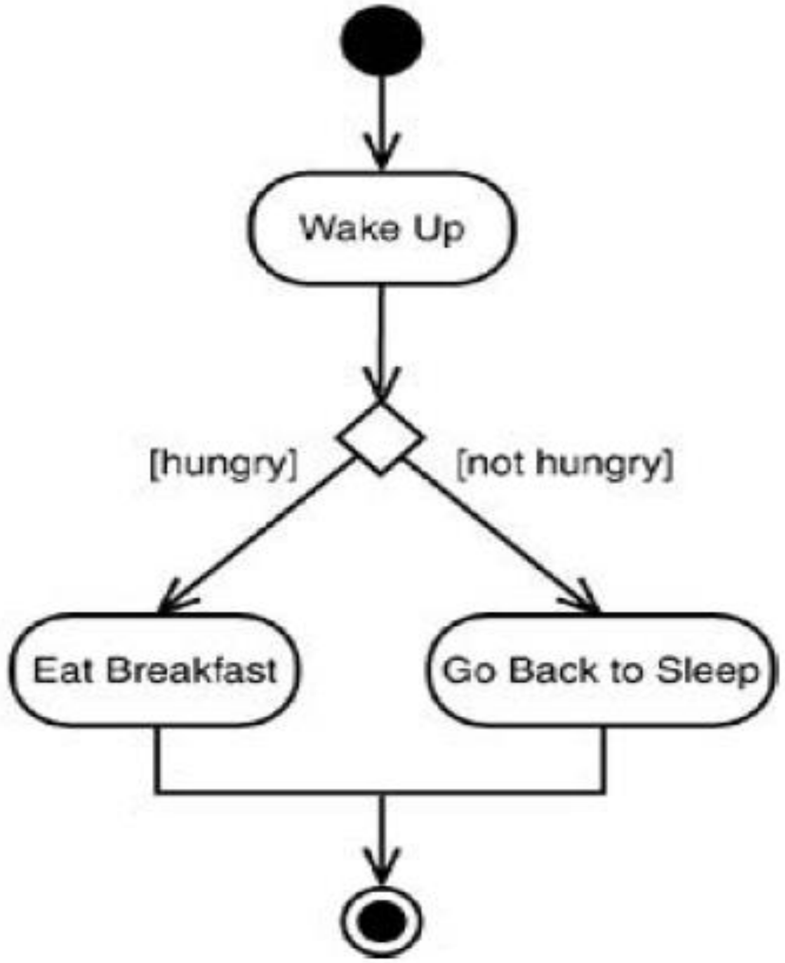


Element	Meaning	Symbol
Activities	Business processes and tasks	▢
Start node	Start of process	●
End node	End of process	◉
Transition	move from one activity to another	→
Synchronization bar	represents **fork** and **join**. Where **fork** represents the concurrent activities, and **join** means end of concurrent activities and is used to move to next sequence activity after finishing concurrent ones.	▬
Decision node	represent alternative ways out of an activity. It has 2 outgoing arcs	◇
Merge node	brings together alternative flow. It has one outgoing flow.	◇
Guard	represents the condition (Boolean test); Guards are associated with transitions. A transition takes place only if its guard evaluates to true at the time the flow of control reaches that transition.	[*condition*]
Swimlane	Swimlanes group **activities** associated with different **roles**. Each swimlane shows who is responsible for which set of related activities.	| |

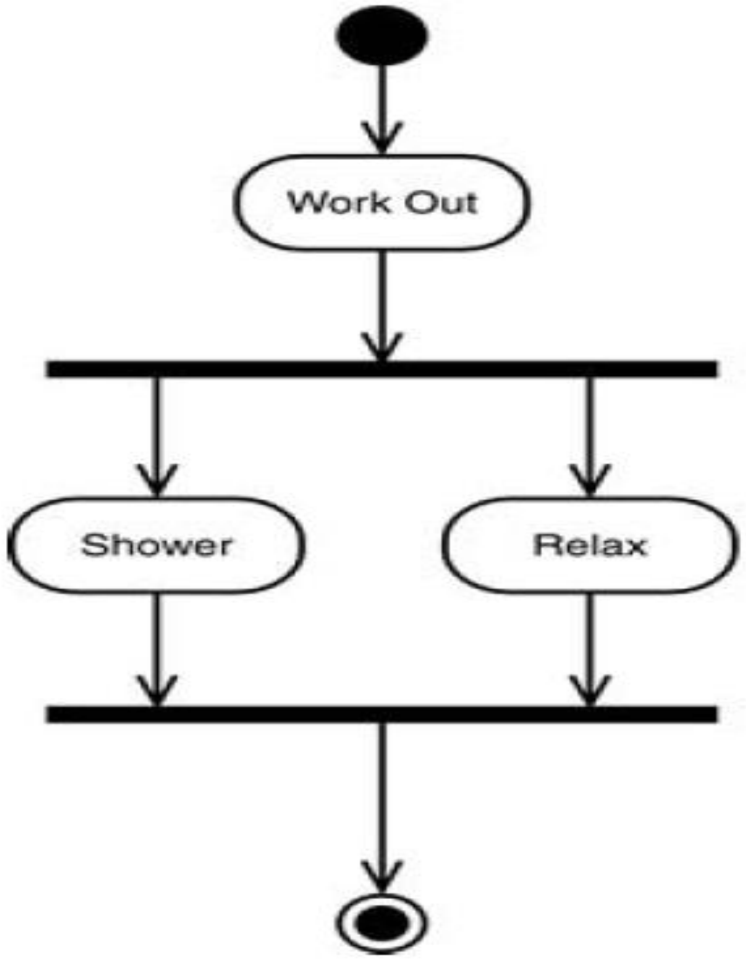
Activity Diagrams: Symbols and Elements



Decisions



Concurrent Paths



Activity Diagrams: Example (1)

1) Suppose, after a hard day’s work, you decide to make yourself a hot drink. What do you do to achieve that goal? What are the actions and how do you organize them?

We start with a **process** called <u>make hot drink</u>. At the center of this task is boiling water in a kettle.

Usually, there are two steps.

1. Fill the kettle with water.
2. Boil the water.

To begin with, we will consider instant coffee (as a particular scenario). Finally, you need a cup to drink from.

Activity Diagrams: Example (1)

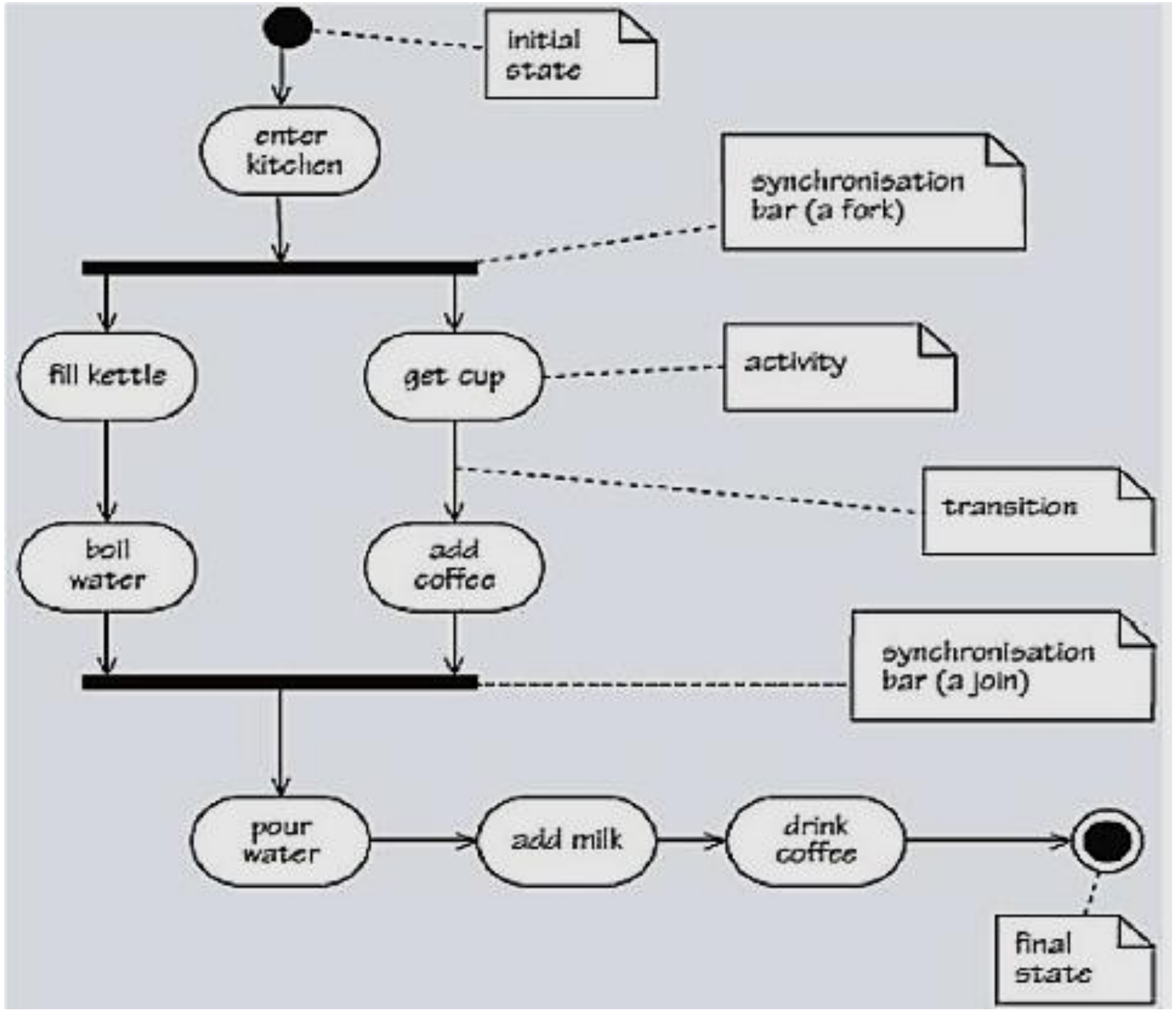


Figure 1

Activity Diagrams: Example (1)

The basic elements of an activity diagram are activities and transitions. The completion of an activity triggers an outgoing transition. Similarly, once the incoming transition has been triggered an activity may commence.

- There are two predefined activities, start and stop, which is represented as filled circles, a diagram may include more than one stop node if it makes it clearer.
- The Figure shows two synchronization bars, denoted by thick horizontal lines, the upper one denotes a fork and the lower one a join. The upper bar allows for separate activities to be carried out in parallel, so between the two synchronization bars the two separate strands can be performed concurrently.
- In UML, a note (with annotation) is shown as a rectangle with the top right-hand corner folded down.
- A dashed line is used to attach the note to the model element to which it refers.

Activity Diagrams: Example (1)

Redraw the activity diagram of preparing a cup coffee so that it allows to check if cup is clean continue the process, if not wash the cup.

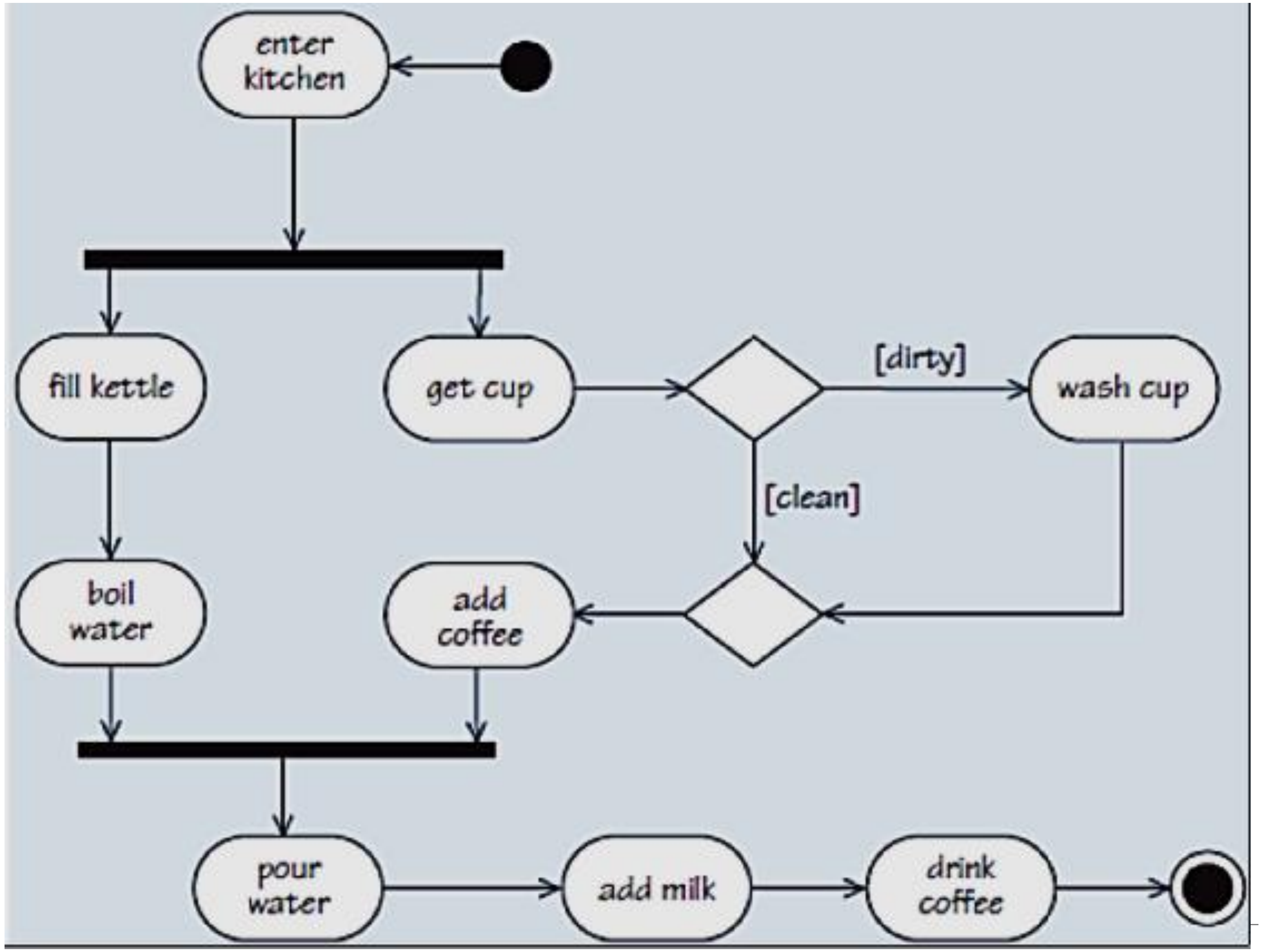


Figure 2

Activity Diagrams: Example (1)

- Each of the transitions leaving the first decision diamond has a guard to determine which path should be taken under a given condition.
- One outgoing transition should always be taken – this means that exactly one of the relevant guards must always evaluate to true.
- A diamond shape is also used as a merge node, which brings together alternative mutually exclusive flows, as shown in figure 2.
- A merge node will be reached only by one of the alternative flows and has a single outgoing flow.
- This is enforced by ‘decision’ and ‘merge’ nodes for alternative outgoing and incoming transitions, and by synchronization bars for concurrent outgoing (fork) and incoming (join) transitions

Activity Diagrams: Example (2)

2) Consider the lending process from a library :

A typical lending library keeps a stock of books for the use of its members. **Each member** can look on the library shelves to select a copy to borrow. Then take out a number of books, up to a certain limit. After a given period of time, the library expects members to return the books that they have on loan.

When borrowing books, a member is expected to wait in queue, then to hand their chosen books to the **librarian**, who records each new loan before issuing the books to the member. After that the librarian will prepare for next member in the queue.

Activity Diagrams: Example (2)

An activity diagram of borrowing a copy of a book from the library.

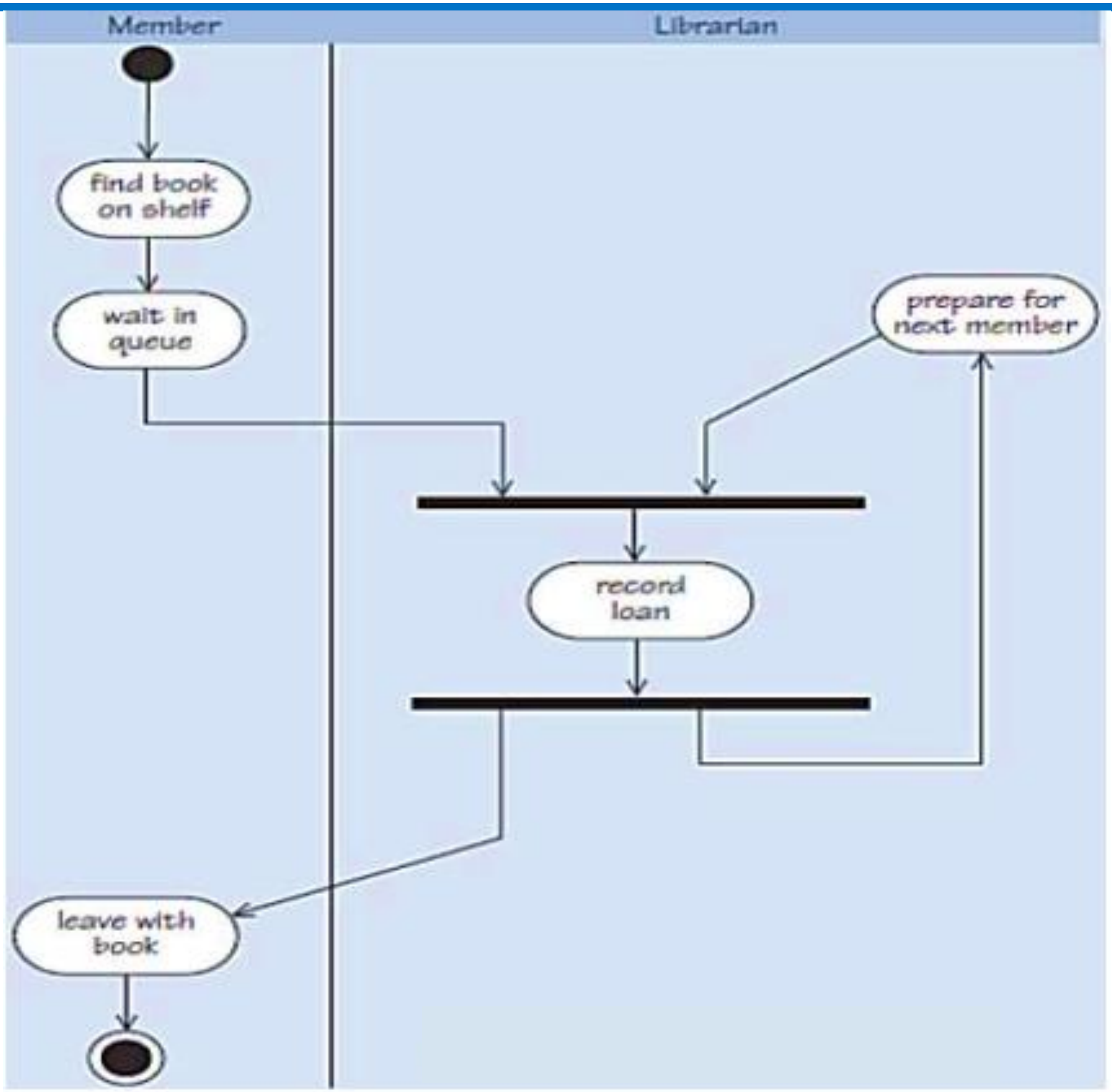


Figure 3

Activity Diagrams: Example (2)

Swim-lanes group activities associated with different roles.

- The swim-lanes show the role that is responsible for each activity.
- Activity diagrams represent the sequence of activities.
- When you are modelling a workflow that involves more than one role, it is possible to identify which role is responsible for a particular activity.
- An activity diagram can help identify the stages at which each role requires some interaction with the process.

In figure 3: The left-hand swim-lane contains the activities that a library member performs when borrowing a book, while the right-hand swim-lane shows the library's self-service system workflow for the same process ('issue copy of book').

Activity Diagrams: Swim lanes

- The activity diagram adds the dimension of visualizing roles.
- To do that, you separate the diagram into parallel segments.
- Each segment shows the name of a role at the top and presents the activities of each role.
- Transitions can take place from one segment to another.

Activity Diagrams: Example (3)

An activity diagram of returning a copy of a book to the library.

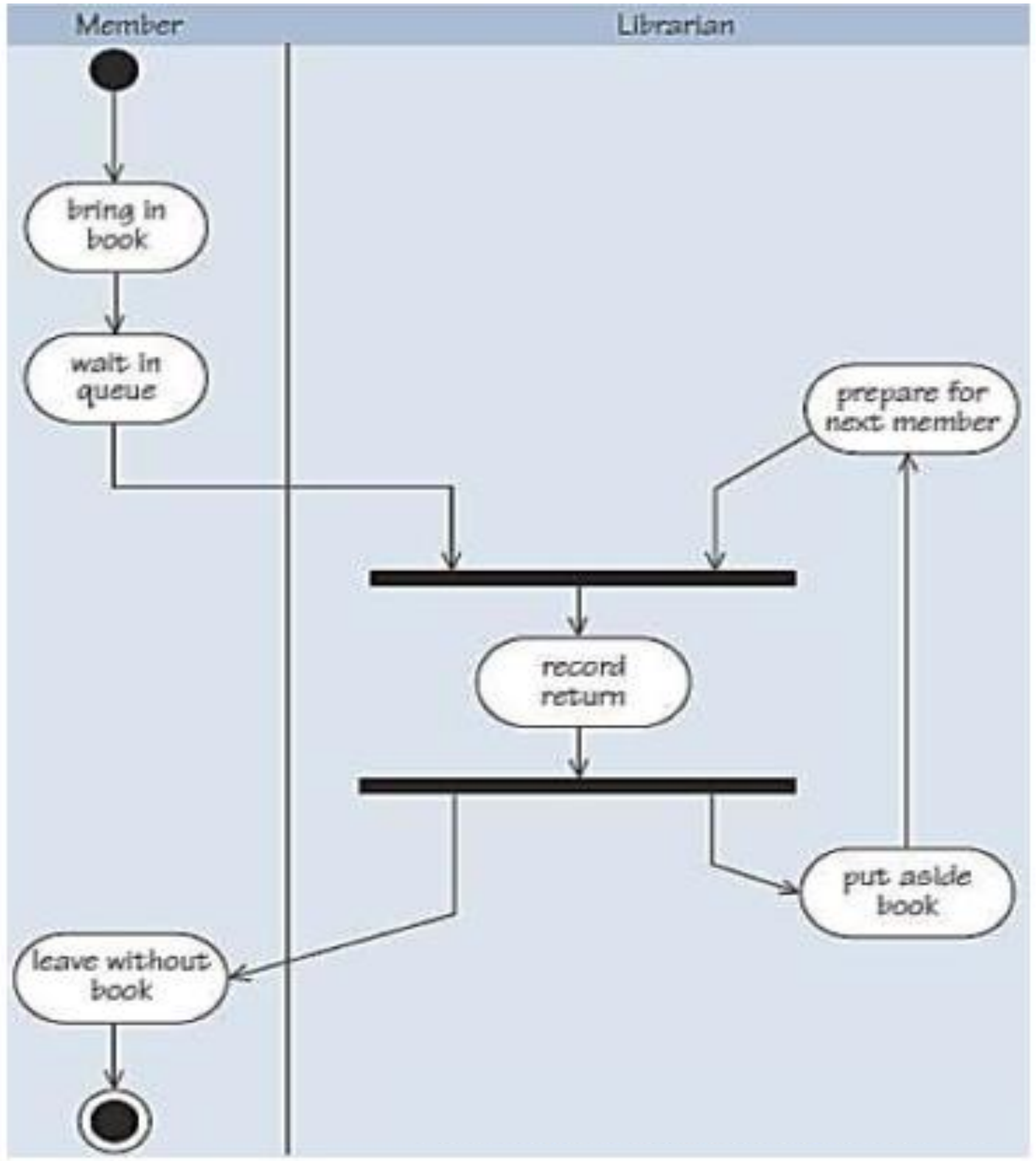
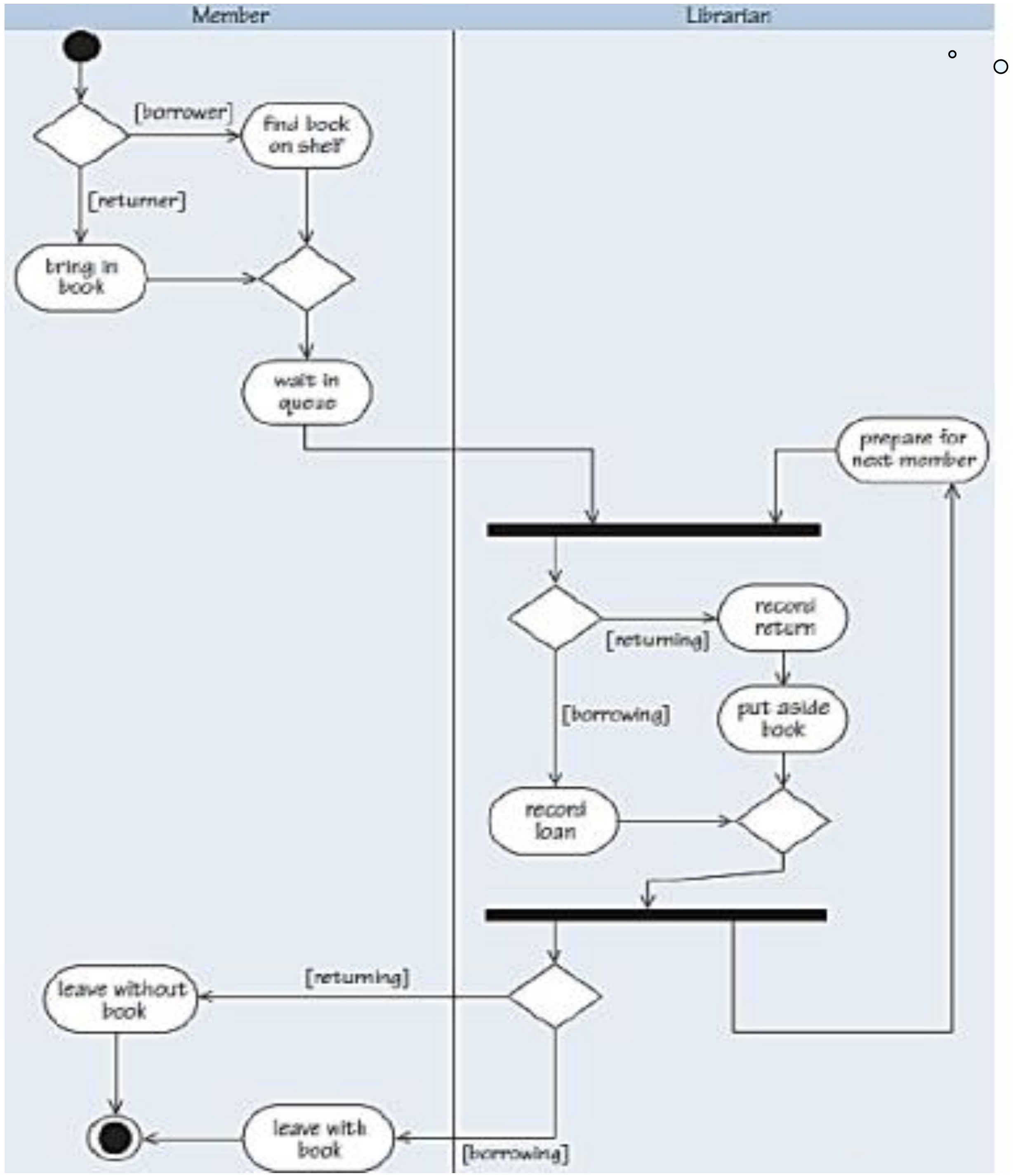


Figure 4

Example (4)

An activity diagram of borrowing and returning a copy of a book.



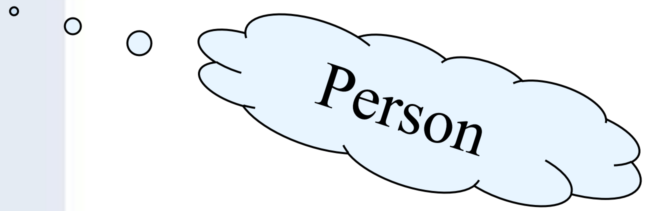


Figure 5

An activity diagram of registering a course

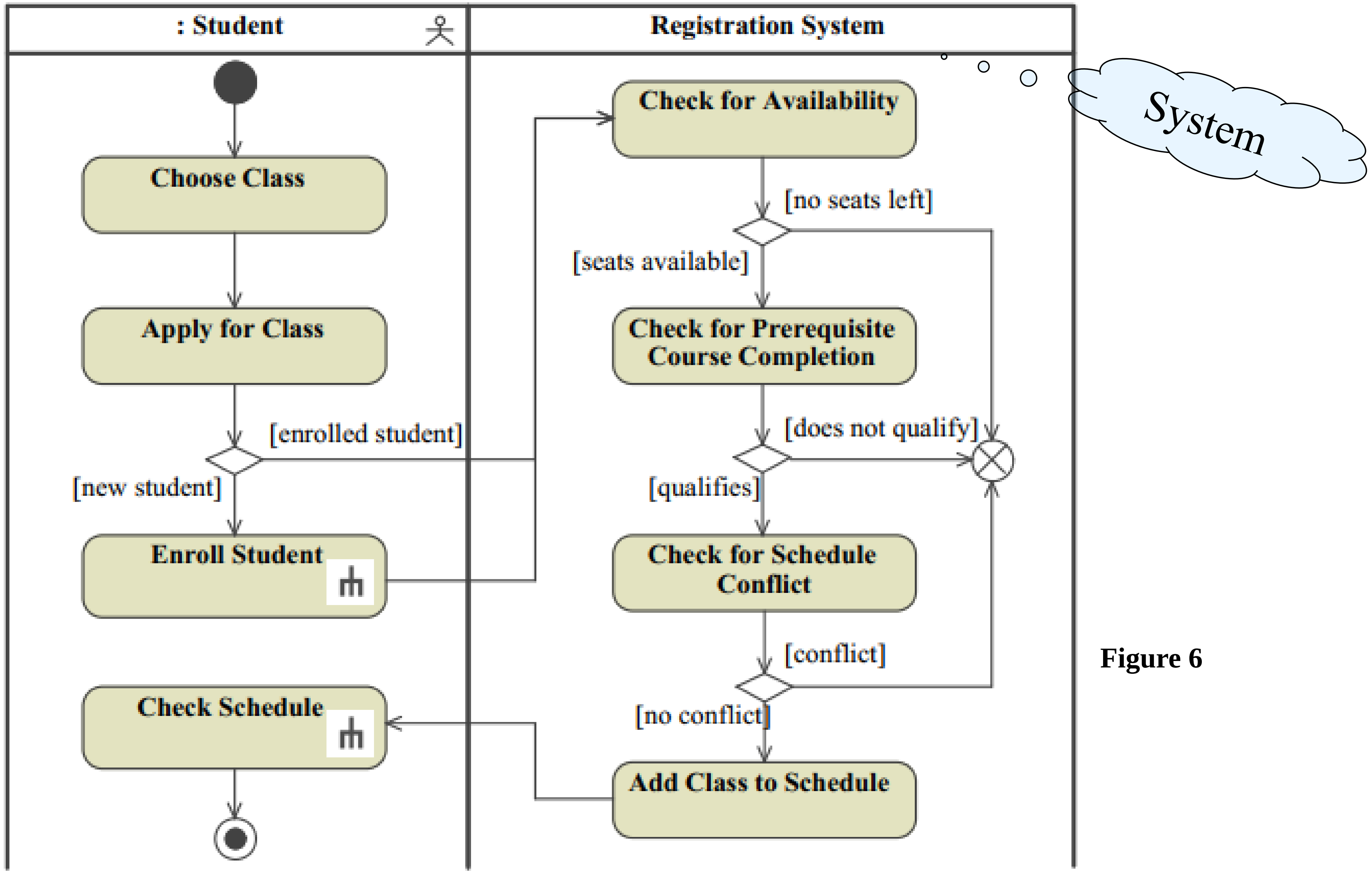
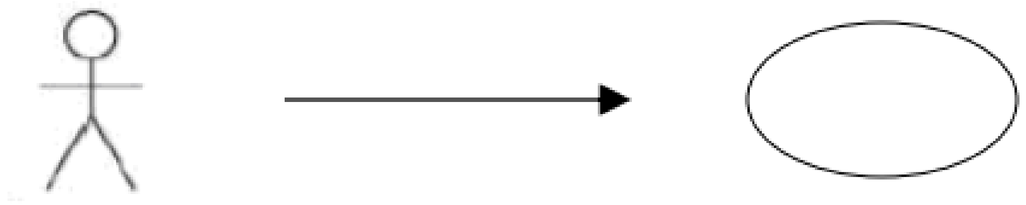


Figure 6

Use Case Modeling

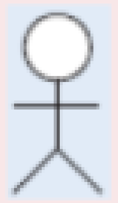
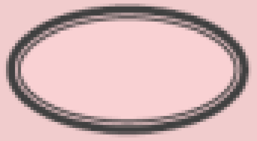
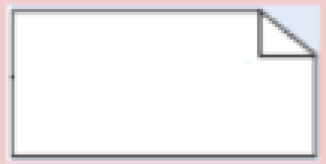
Use Case Modeling

- Use case modeling is widely used to support **requirements elicitation** because it describes what a user expects from a system.
- Each use case represents a discrete task **(functionality)** that involves external interaction with a system.
- It displays the **relationship** among **actors** and **use cases**:
  - **Use cases:** what the system should do.
  - **Actors:** anyone or anything outside the system's scope but interacts with it with the system (Individual, group, company,…).



Use Case Diagram: Elements & Symbols

Introduces a means of defining what a proposed system should do from a **user's perspective**.

The actors, represented by **stick figures**;	
The use cases, represented by **ovals**, and are used to represent tasks.;	
The relationships between actors and use cases, represented by **lines**.	────────
Note symbol: is used to clarify an aspect or a task, it can be used with any UML diagram.	
A **dashed line** is used to attach the note to the model element to which it refers.	‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒

Use Case Descriptions

- Use Case diagrams aren't intended to show in which **order** the use cases are executed.
- Use cases are an **implementation-independent**
  - High-level view of what the user expects from the system
  - Focus on what the system should do, not how the system will
- Use case diagrams give a fairly simple overview of an interaction, so you have to provide more detail to understand what is involved. This detail can either be:
  - A simple textual description (flow of events)
  - A structured description in a **table**.
  - A sequence diagram

Use Case Descriptions

Each use case description should contain the following parts in minimum:

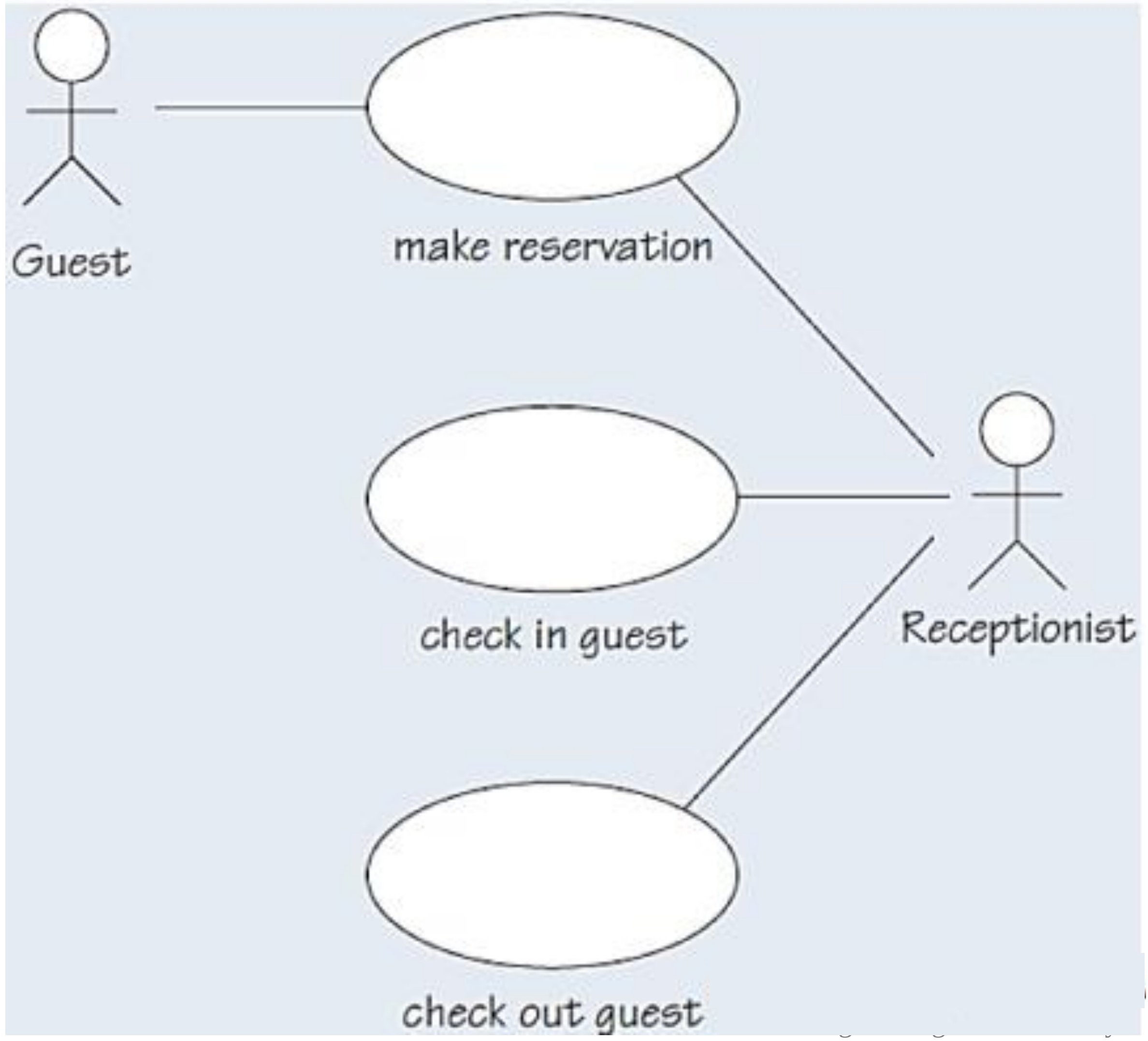
- **Use case:** identifier and name.
- **Initiator:** name of the actor who initiates the process.
- **Pre-condition(s):** a condition that must hold before this use case can be carried out.
- **Post-condition(s):** a condition that must hold after the use case has been completed.
- **Main Success Scenario:** a single sequence of steps that describe the main success scenario. You can number the steps. A scenario is an instance of a use case.
- **Goal:** a short description of the goal of the use case;

Use Case Scenarios

- A scenario describes a sequence of interactions between the system and some actors. For each use case there is a set of possible *scenarios*. A scenario is an instance of a use case. Here are two examples of scenarios:
    - A member of a lending library wishes to borrow a book, and is allowed to do that as long as they have no outstanding loans.
    - Another member wishes to borrow a book, but has exceeded the quota for the number of books that can be borrowed.
- In each scenario the member wishes to borrow a book, but both the circumstances and outcomes of events are different in each instance.
- The main **success scenario** shows the steps normally followed to achieve the stated goal of the use case.

Use Case Diagrams Example

A simple use case diagram for the main tasks in a hotel system: making reservations, checking in, and checking out, where the reservation can be done by the receptionist or a guest (by Phone/Web).



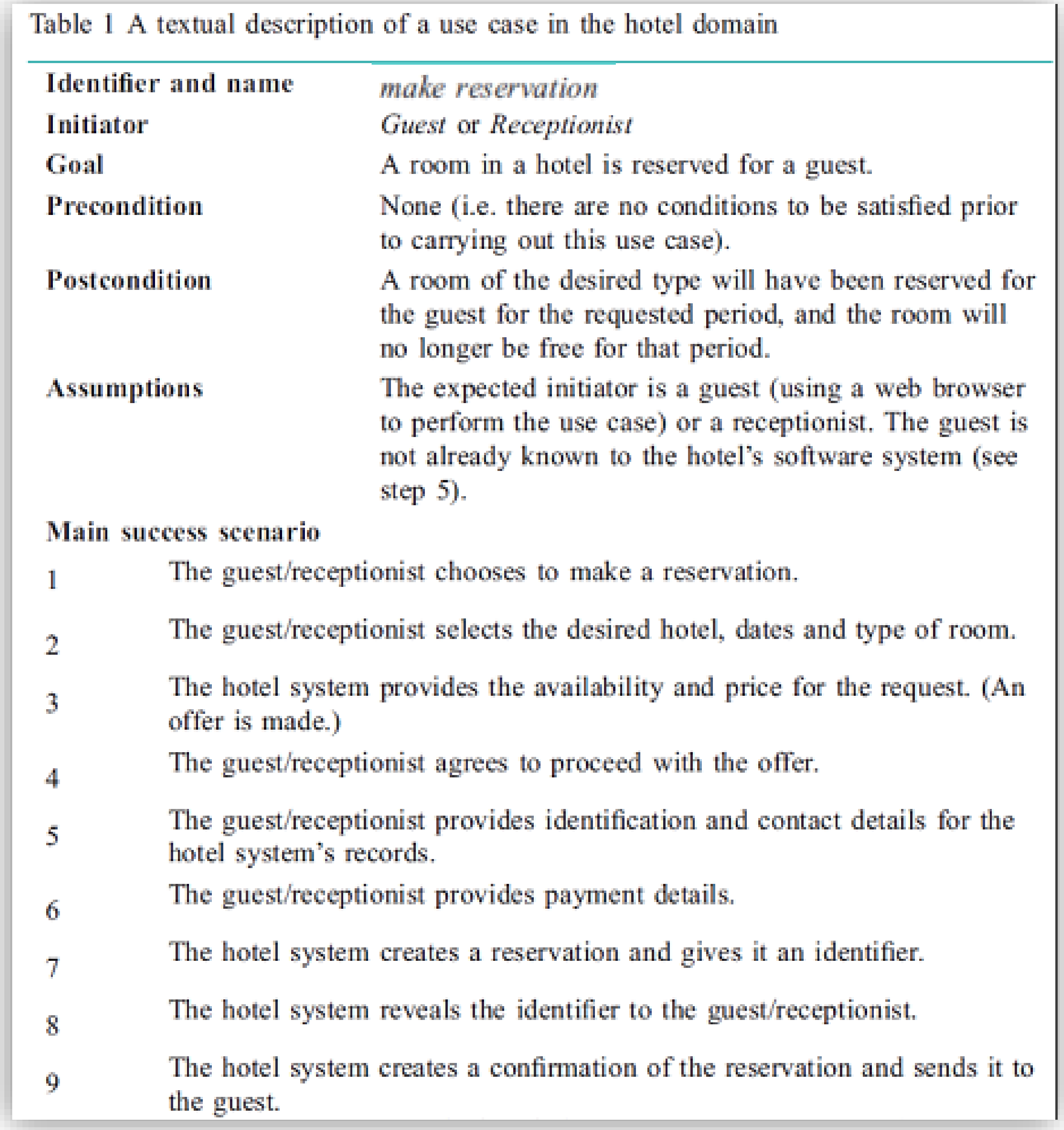
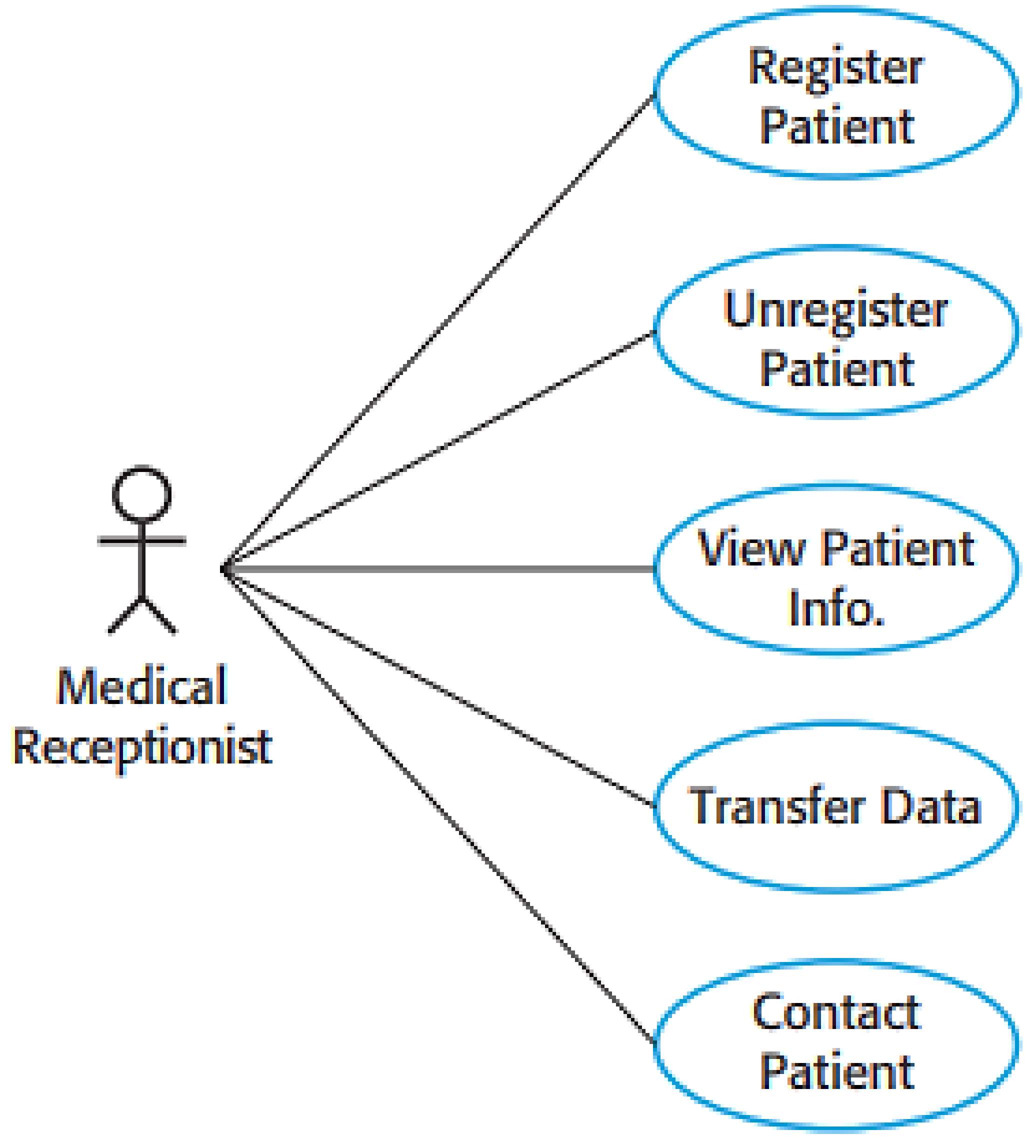


Table 1 A textual description of a use case in the hotel domain

Identifier and name	*make reservation*
Initiator	*Guest* or *Receptionist*
Goal	A room in a hotel is reserved for a guest.
Precondition	None (i.e. there are no conditions to be satisfied prior to carrying out this use case).
Postcondition	A room of the desired type will have been reserved for the guest for the requested period, and the room will no longer be free for that period.
Assumptions	The expected initiator is a guest (using a web browser to perform the use case) or a receptionist. The guest is not already known to the hotel's software system (see step 5).
Main success scenario	
1	The guest/receptionist chooses to make a reservation.
2	The guest/receptionist selects the desired hotel, dates and type of room.
3	The hotel system provides the availability and price for the request. (An offer is made.)
4	The guest/receptionist agrees to proceed with the offer.
5	The guest/receptionist provides identification and contact details for the hotel system's records.
6	The guest/receptionist provides payment details.
7	The hotel system creates a reservation and gives it an identifier.
8	The hotel system reveals the identifier to the guest/receptionist.
9	The hotel system creates a confirmation of the reservation and sends it to the guest.

Use Case Diagrams Example



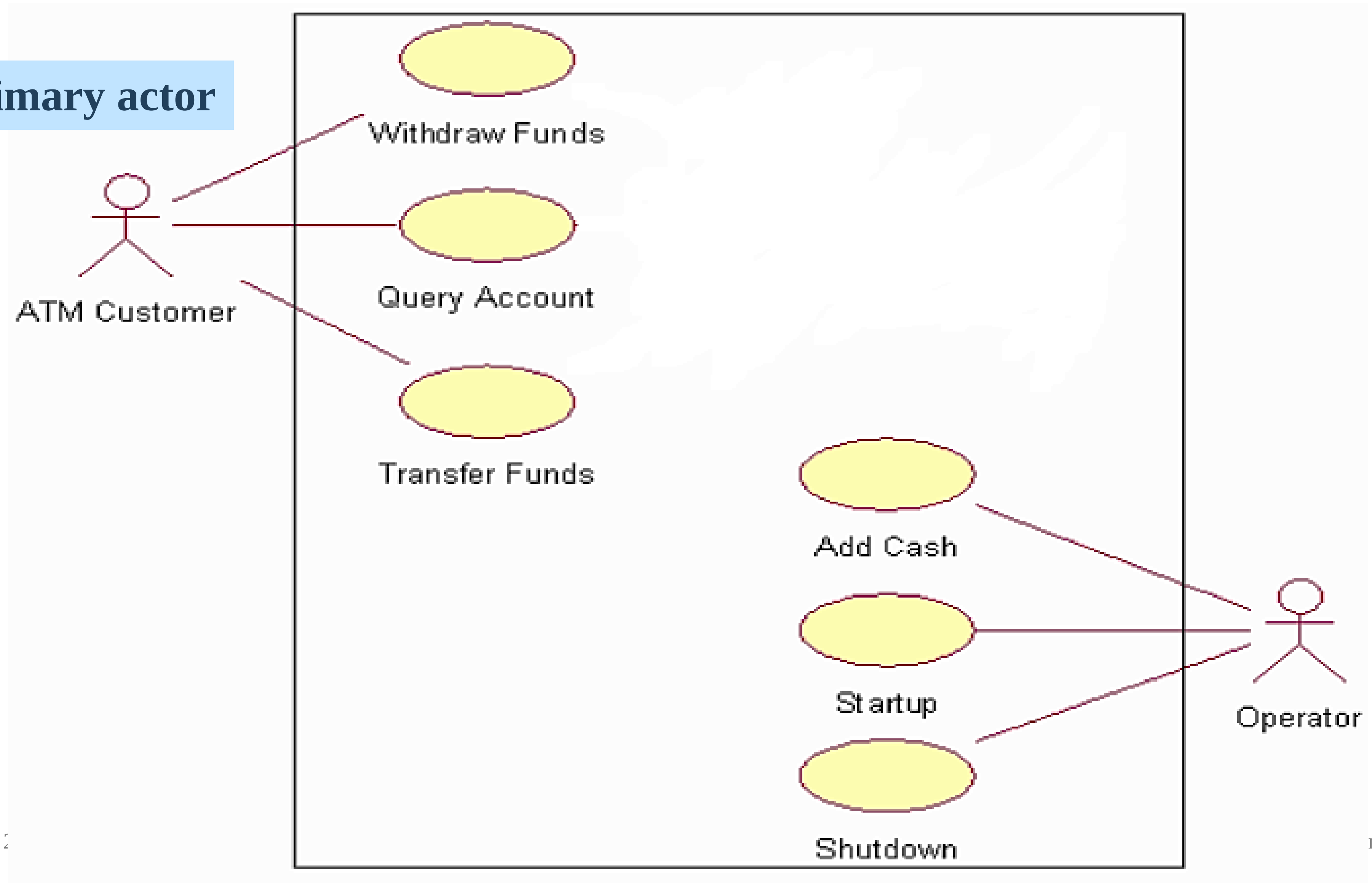
Use Case Diagrams System Boundary

The **system boundary** determines the scope of the system.

- It distinguishes between internal and external components.
- The *external* components are *actors* and the *internal* components are the *use cases*.
- In UML it is <u>optional</u> to use system boundary notation.
- A solid box drawn around the use cases with the actors located outside it represents the system boundary.
- **We use system boundary notation when the system is complex and includes several subsystems.**

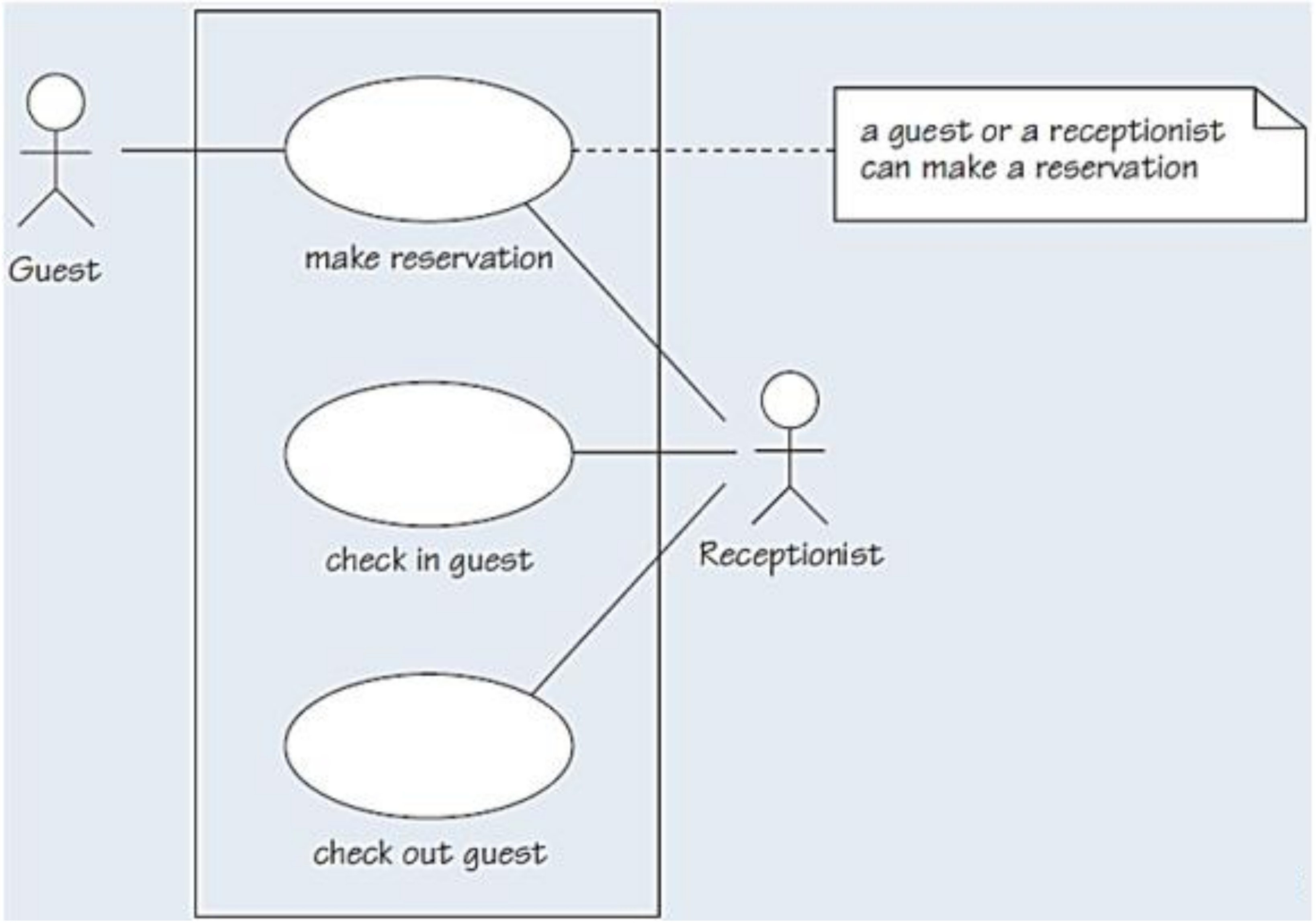
Use Case Diagrams Example

Primary actor



Secondary actor

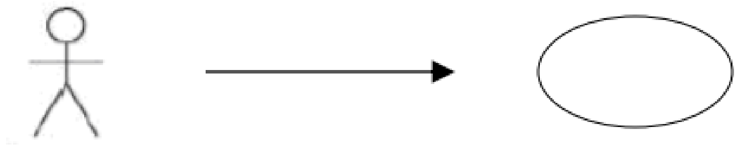
Use Case Diagrams Example



Relationships

1. Association

Used to show the relationship between a use case and an actor only



2. Inclusion
3. Extension

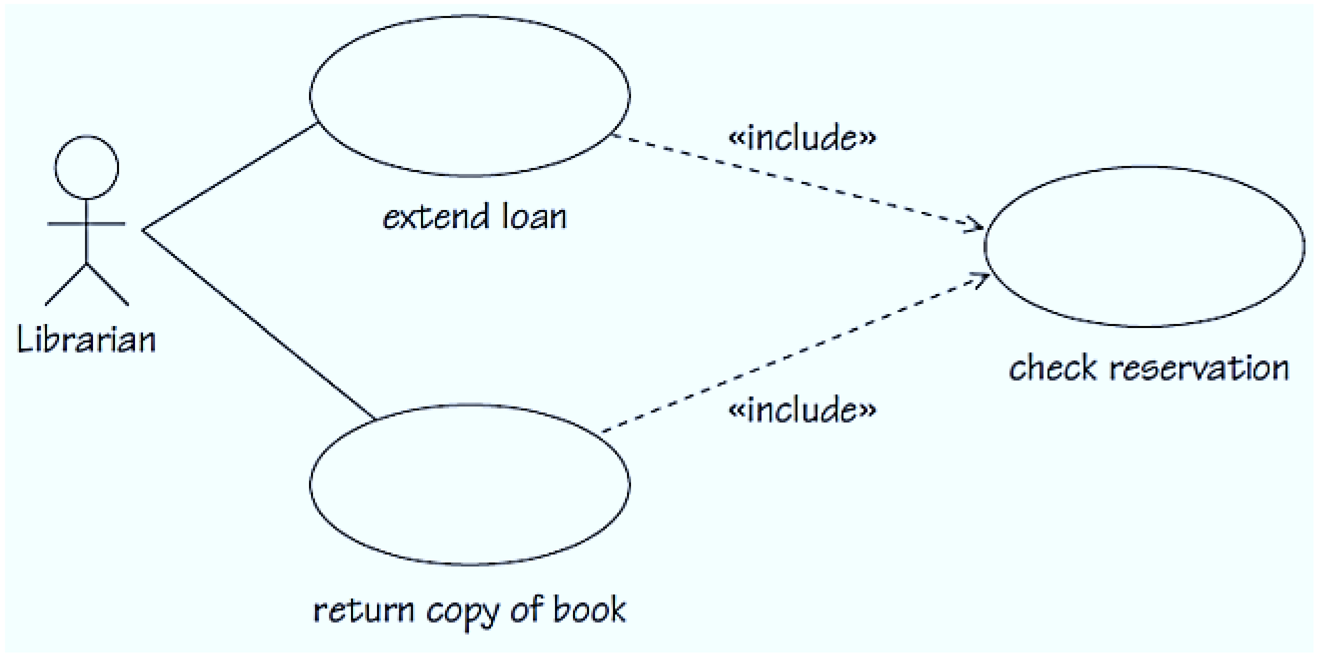
Only two relationships allowed between use cases

4. Generalization

Between actors, and sometimes between use cases

Inclusion Relationship

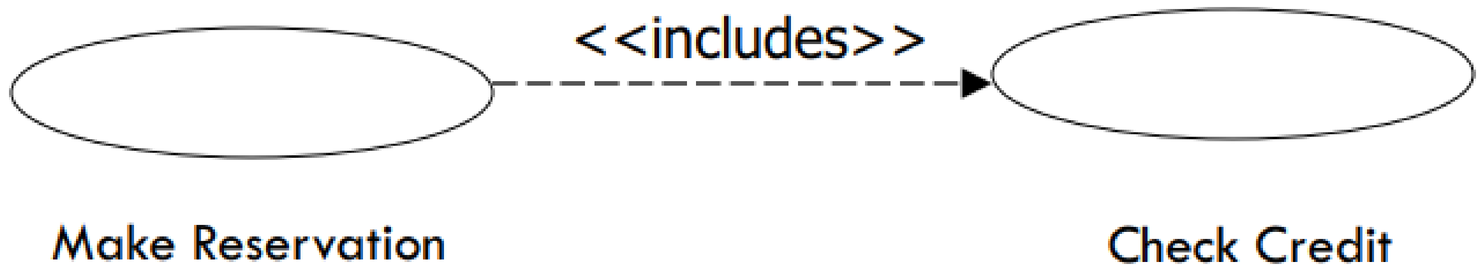
- This is when two or more use cases have a **common** area of functionality that can be factored out as a distinct use case.
- An includes relationship suggests that one use case **always** uses the functionality provided by another.



In UML we use **«include»** for inclusion

Inclusion Relationship

- In the process of eliciting and specifying requirements, you may find a certain amount of **common behavior** in two or more of your use cases.
  - You can record the shared behavior **in a new use case** and connect it to the use cases that it came from with a dashed arrow (indicating a dependency relationship) pointing from the original use case to the new one.
  - Hence the dependency arrow is labeled with the «include» stereotype.



Exclusion Relationship

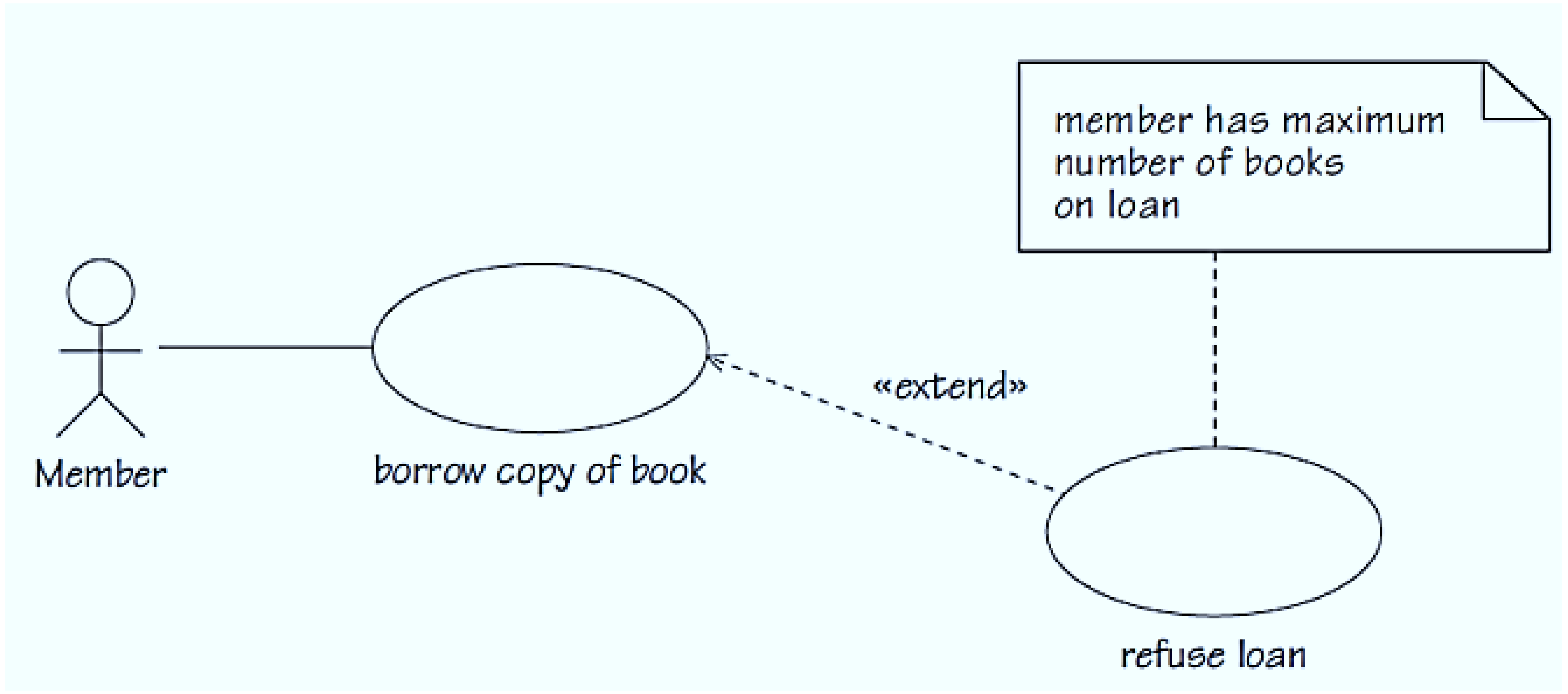
- Extends relationship allows one use case **(extending)** to extend the functionality provided by another use case **(base)**. As the name implies it extends the base use case and adds more functionality to the system.

Few things to consider when using the **«extend»** relationship:

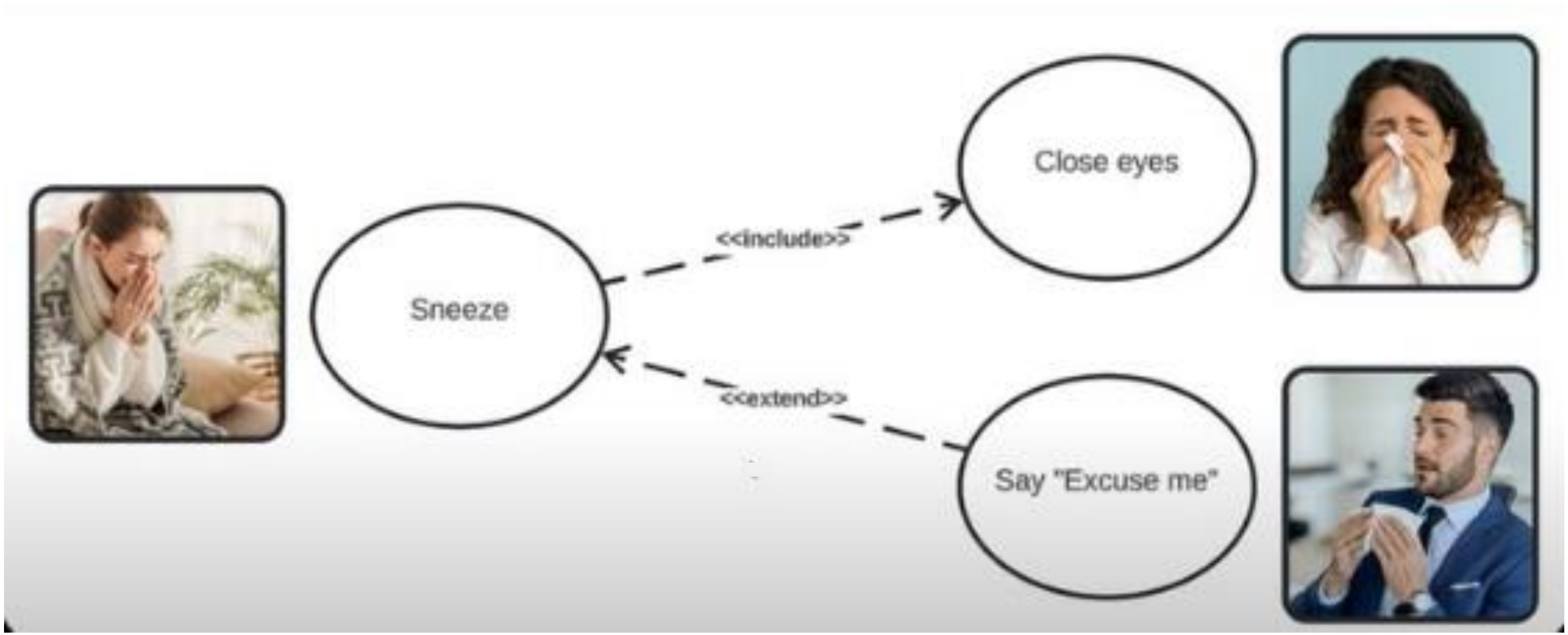
  - The extending use case is dependent on the extended (base) use case.
  - The extending use case is usually **optional**.
  - The extended (base) use case must be meaningful on its own.

- You should add a **condition** to each extension – by a note to specify when the variant behavior will be included.

Exclusion Relationship

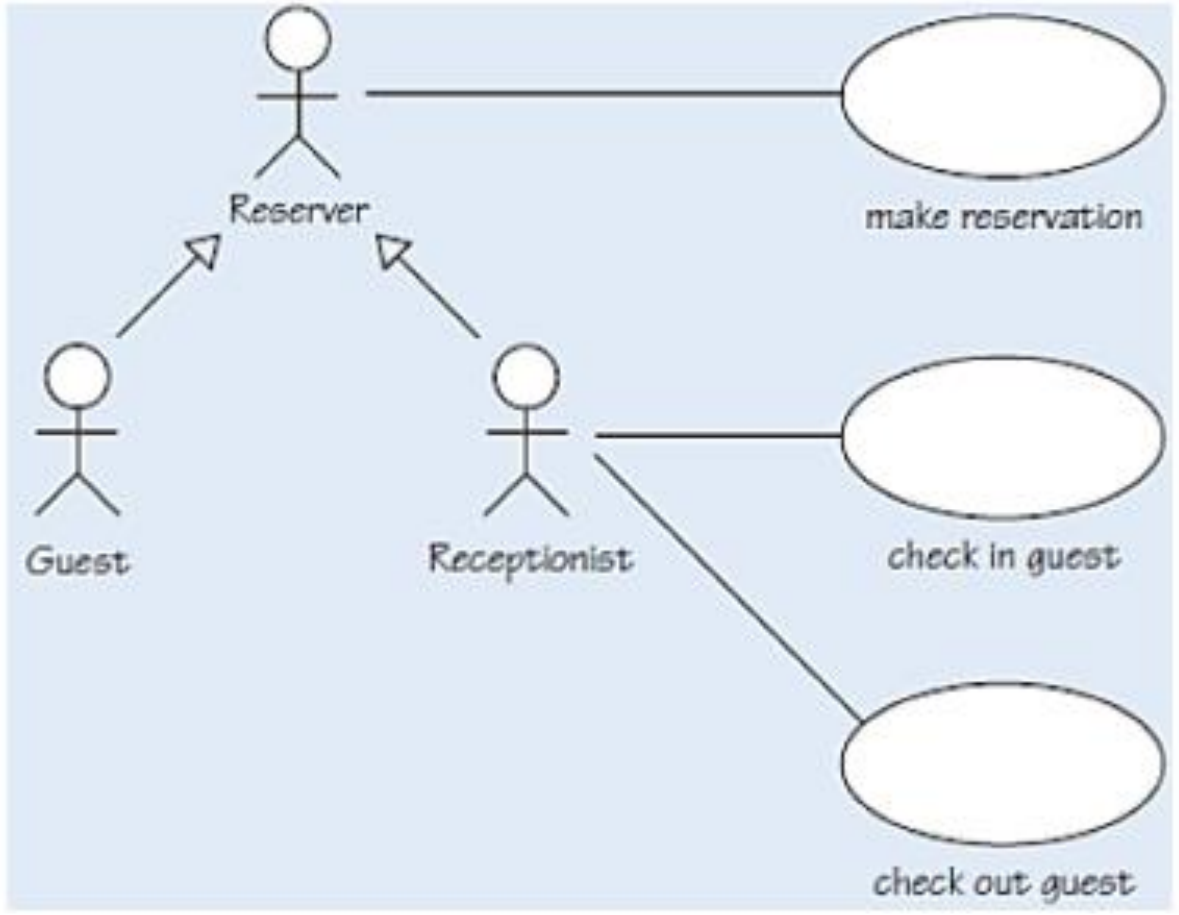


Inclusion & Exclusion Relationship

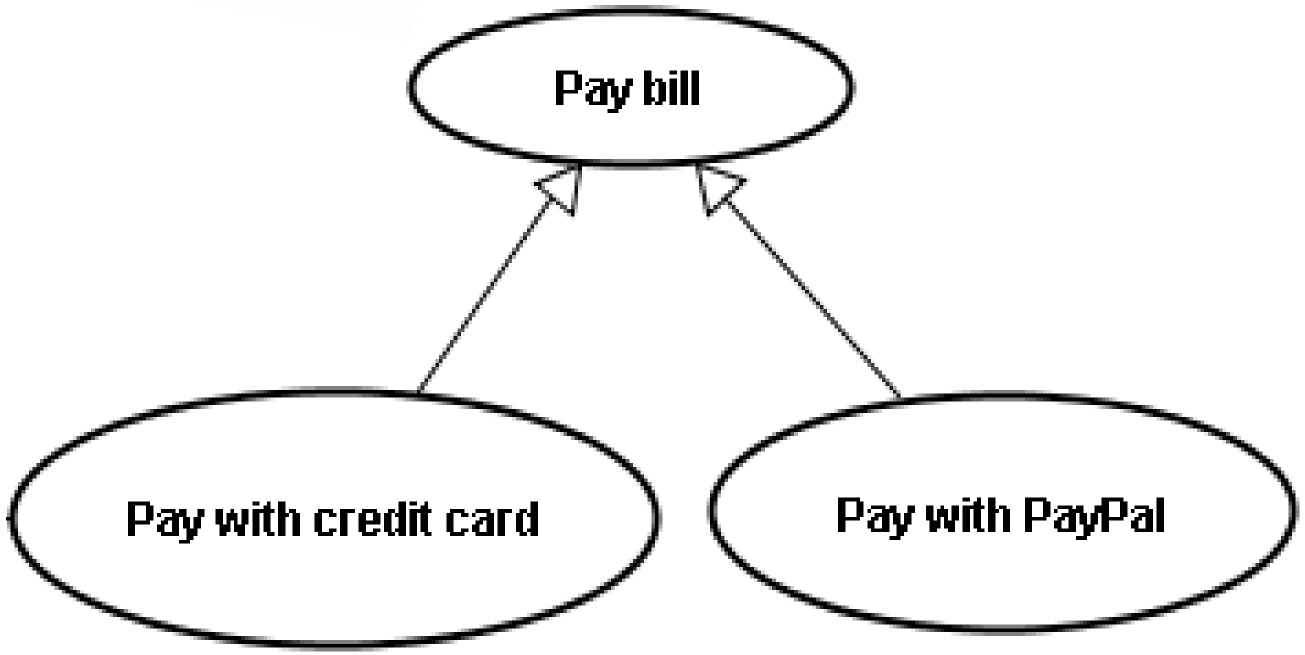


Generalization Relationship (Actors)

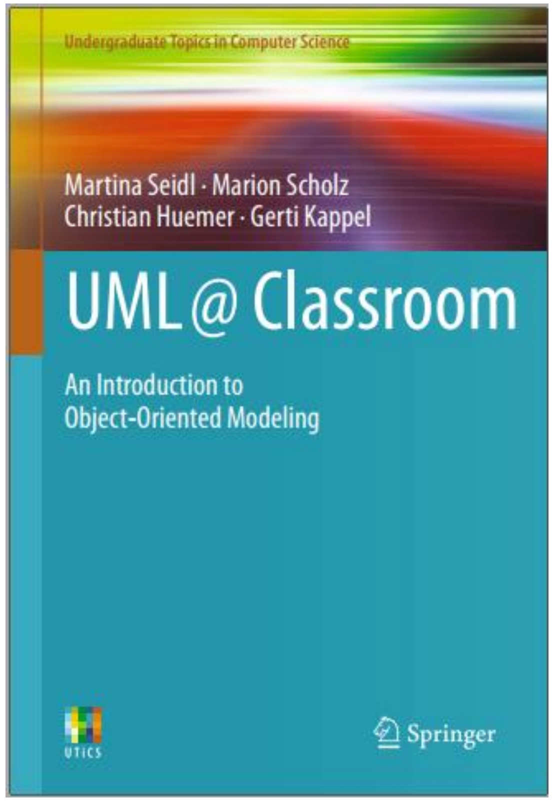
- UML provides a notation to use generalization between actors.
  - When two actors share the same behavior (interacting with the same use cases) and one of them has some extra behavior, then the common behavior can be associated with a generalized actor and the more specific behavior with the specialized actor.



Generalization Relationship (Use-cases)



Additional References


Undergraduate Topics in Computer Science

Martina Seidl · Marion Scholz
Christian Huemer · Gerti Kappel

UML @ Classroom

An Introduction to
Object-Oriented Modeling

UTiCS

Springer

Thank You