Modeling Interactions and Behavior

Dr. Marouane Kessentini
Department of Computer Science

Interaction Diagrams

- Interaction diagrams are used to model the dynamic aspects of a software system
  - They help you to visualize how the system runs.
  - An interaction diagram is often built from a use case and a class diagram.
    - The objective is to show how a set of objects accomplish the required interactions with an actor.

Interactions and messages

- Interaction diagrams show how a set of actors and objects communicate with each other to perform:
  - The steps of a use case, or
  - The steps of some other piece of functionality.
- The set of steps, taken together, is called an *interaction*.
- Interaction diagrams can show several different types of communication.
  - E.g. method calls, messages send over the network
  - These are all referred to as *messages*.

Elements found in interaction diagrams

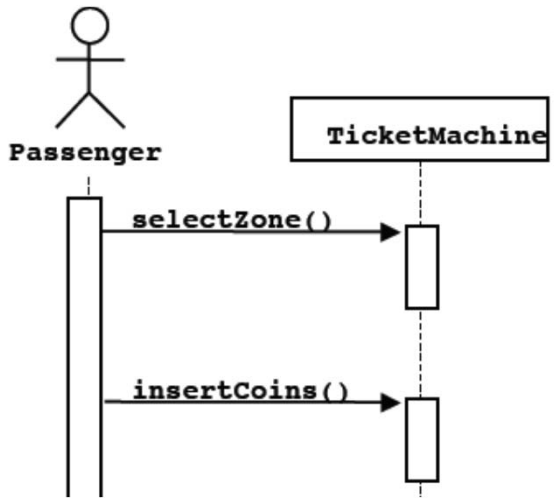
- Instances of classes
  - Shown as boxes with the class and object identifier underlined
- Actors
  - Use the stick-person symbol as in use case diagrams
- Messages
  - Shown as arrows from actor to object, or from object to object

Creating interaction diagrams

- You should develop a class diagram and a use case model before starting to create an interaction diagram.
  - There are two kinds of interaction diagrams:
    - *Sequence diagrams*
    - *Communication diagrams*

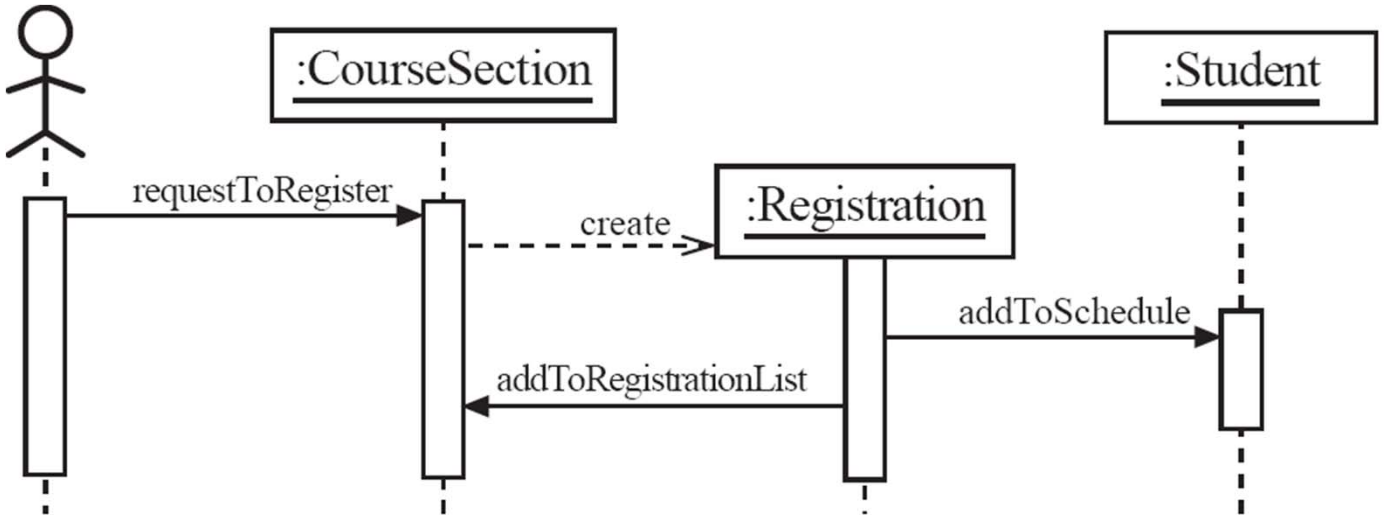
Sequence Diagrams

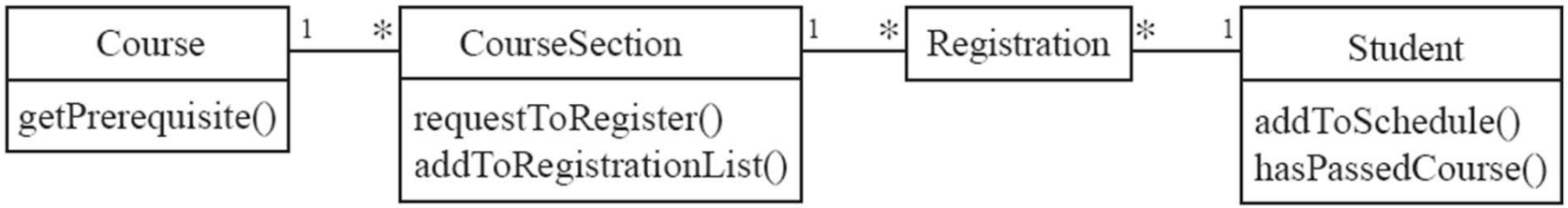
- Used during requirements analysis
  - To refine use case descriptions
  - to find additional objects ("participating objects")
- Used during system design
  - to refine subsystem interfaces
- Used during Testing
  - to specify expected behavior and validate output



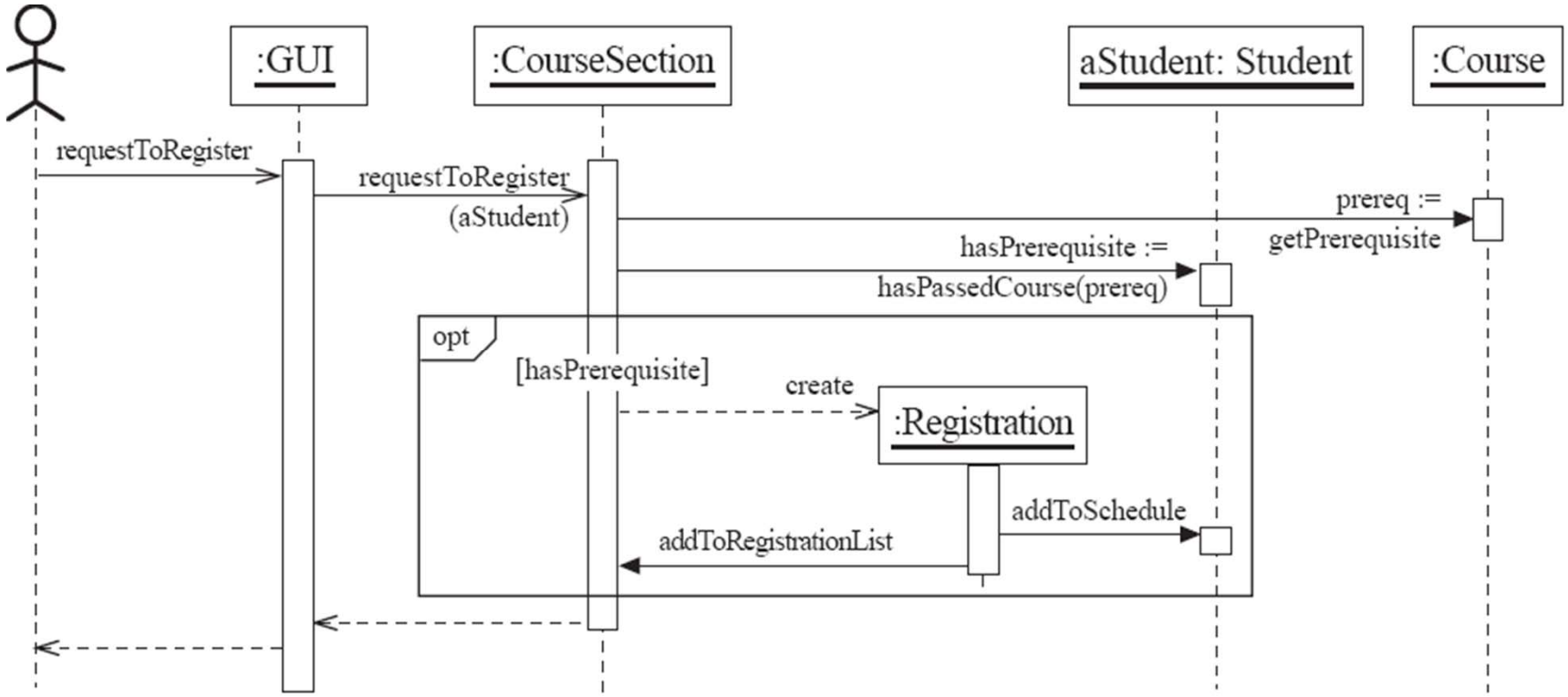
- Classes are represented by rectangles
- Lifelines are represented by dashed lines
- Messages are represented by arrows
- Activations are represented by narrow rectangles.

Sequence diagrams – an example



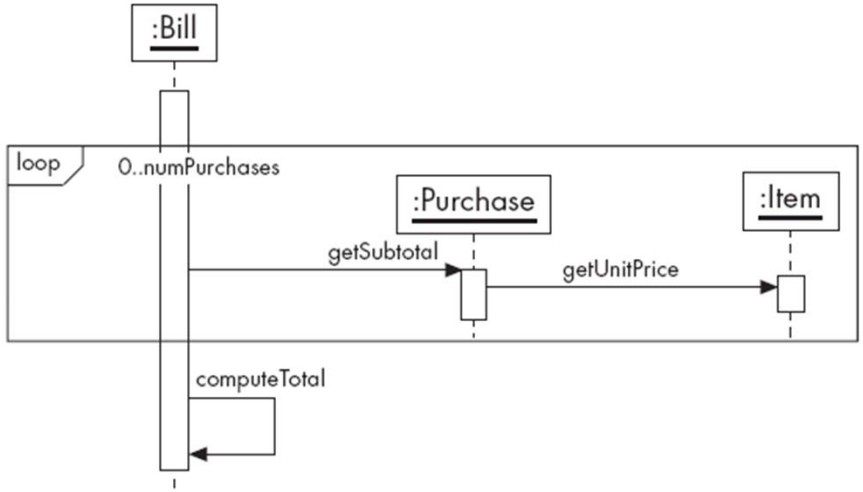


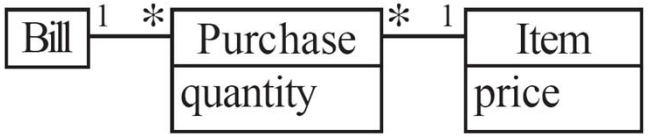
Sequence diagrams – same example, more details



Sequence diagrams – an example with replicated messages

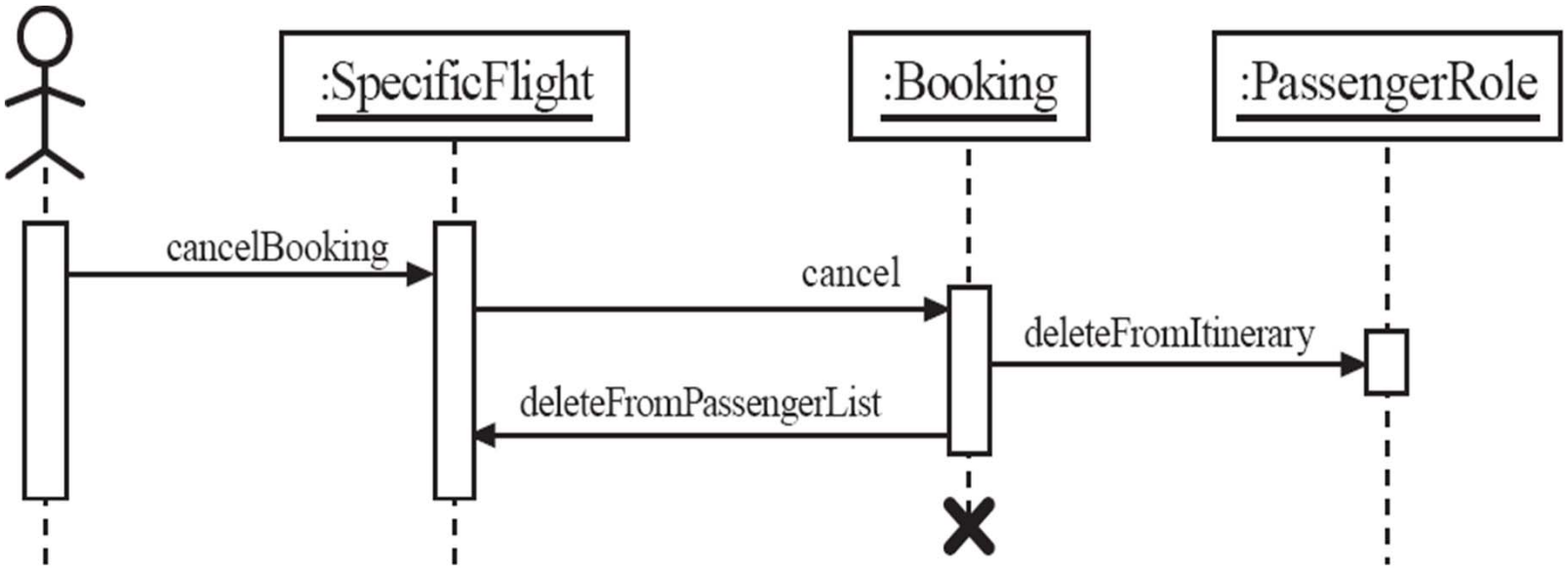
– An *iteration* over objects is indicated by an asterisk preceding the message name



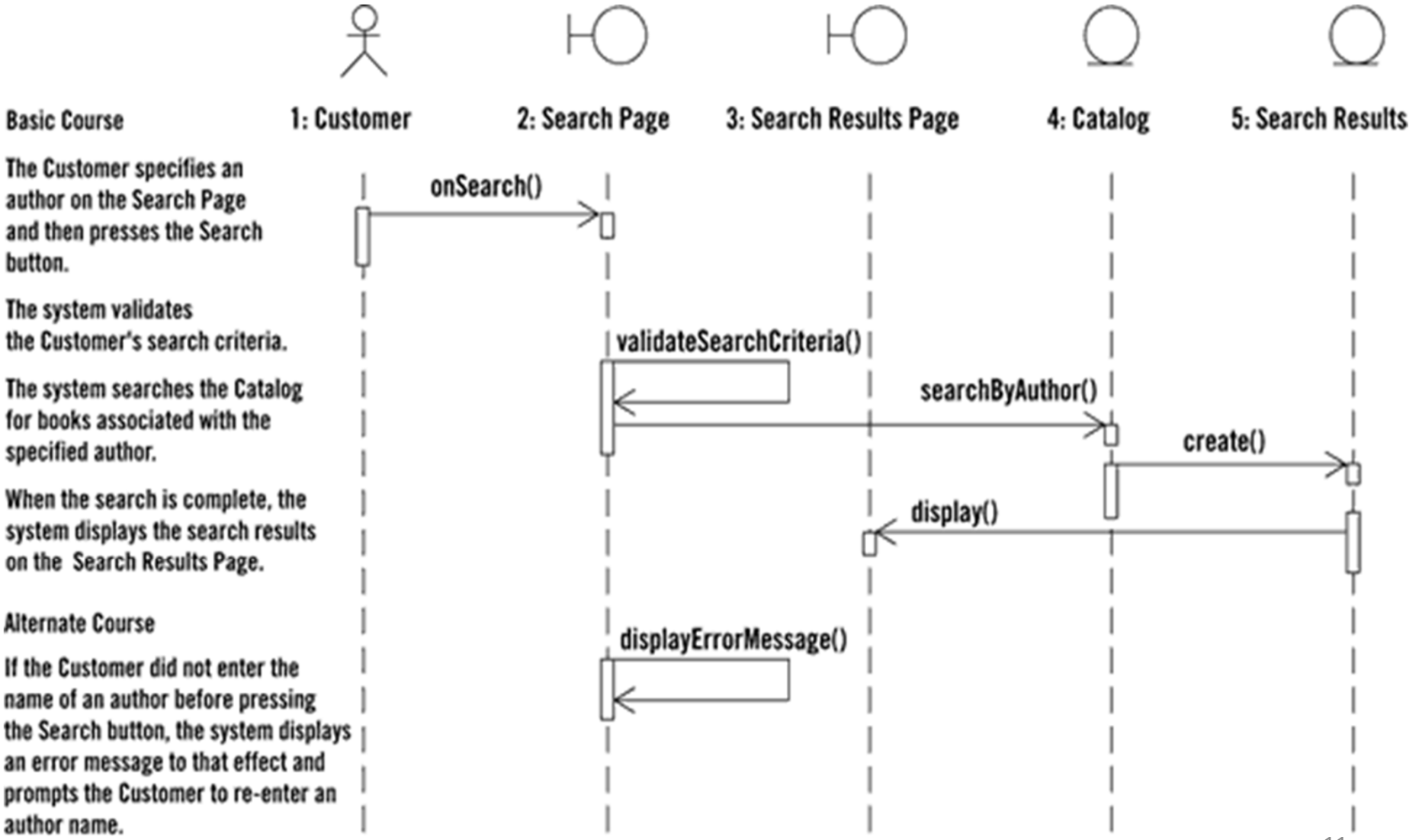


Sequence diagrams – an example with object deletion

– If an object's life ends, this is shown with an X at the end of the lifeline



SD From Use Case



State Diagrams

- A state diagram describes the behaviour of a *system*, some *part* of a system, or an *individual object*.
  - At any given point in time, the system or object is in a certain state.
    - Being in a state means that it is will behave in a *specific way* in response to any events that occur.
  - Some events will cause the system to change state.
    - In the new state, the system will behave in a different way to events.
  - A state diagram is a directed graph where the nodes are states and the arcs are transitions.

States

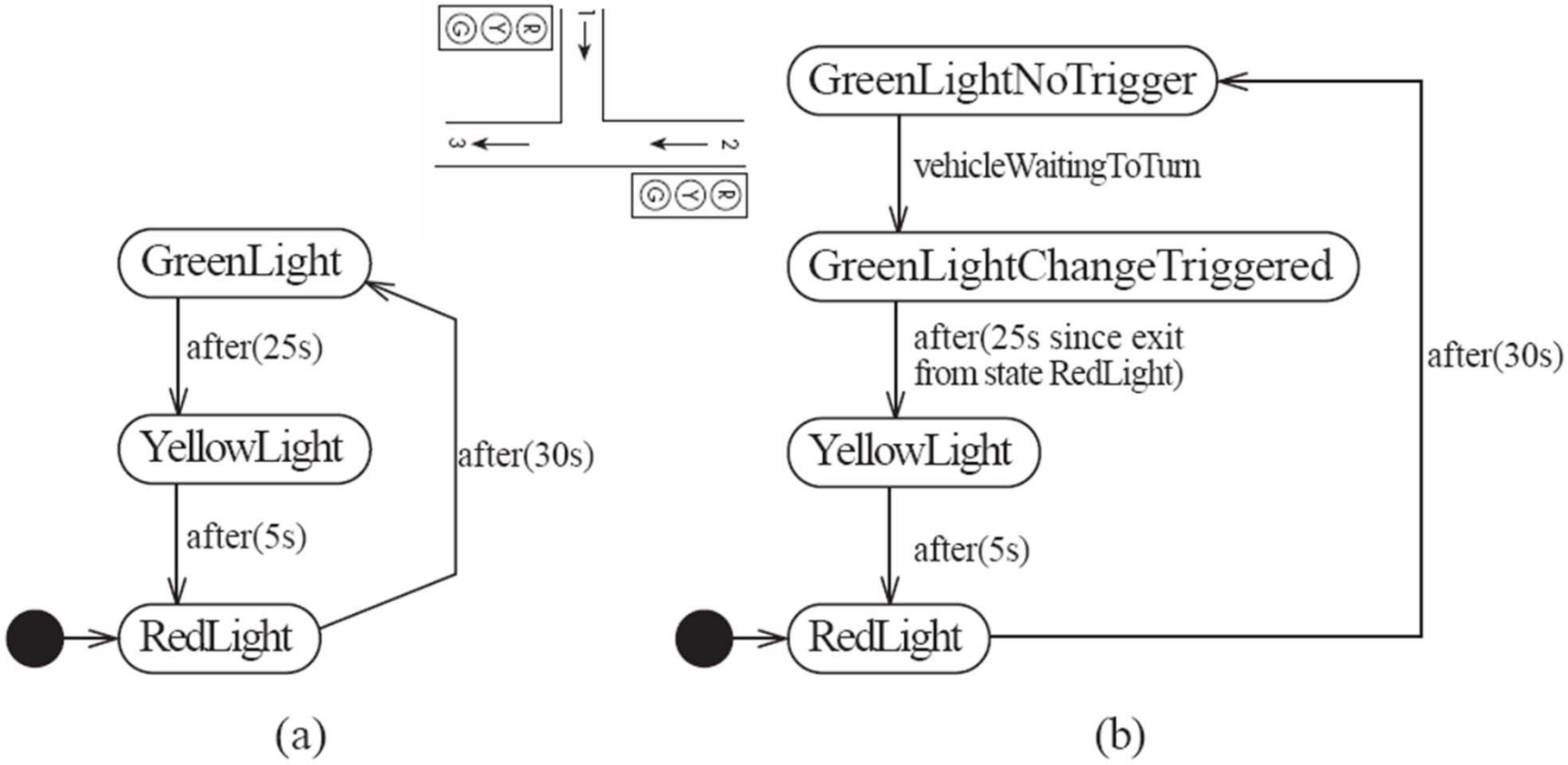
- At any given point in time, the system is in one state.
- It will remain in this state until an event occurs that causes it to change state.
- A state is represented by a rounded rectangle containing the name of the state.
- Special states:
  - A black circle represents the *start state*
  - A circle with a ring around it represents an *end state*

Transitions

- A transition represents a change of state in response to an event.
  - It is considered to occur instantaneously.

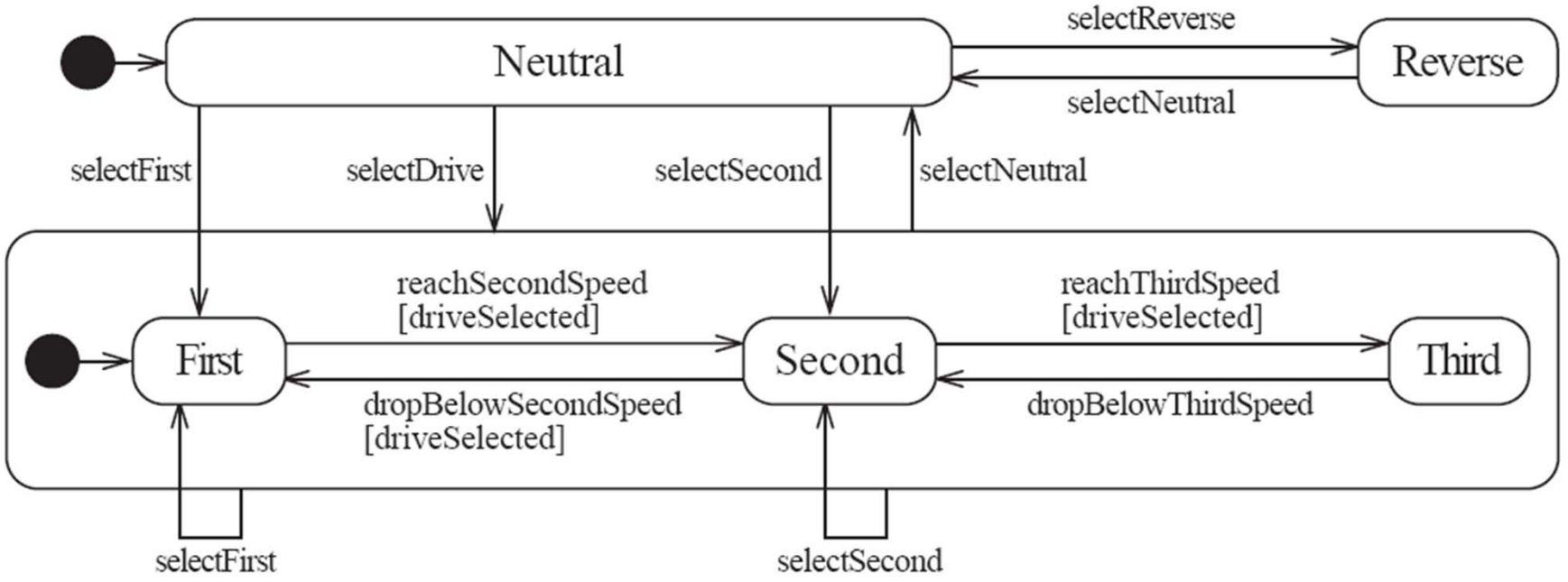
- The label on each transition is the event that causes the change of state.

State diagrams – an example of transitions with time-outs and conditions

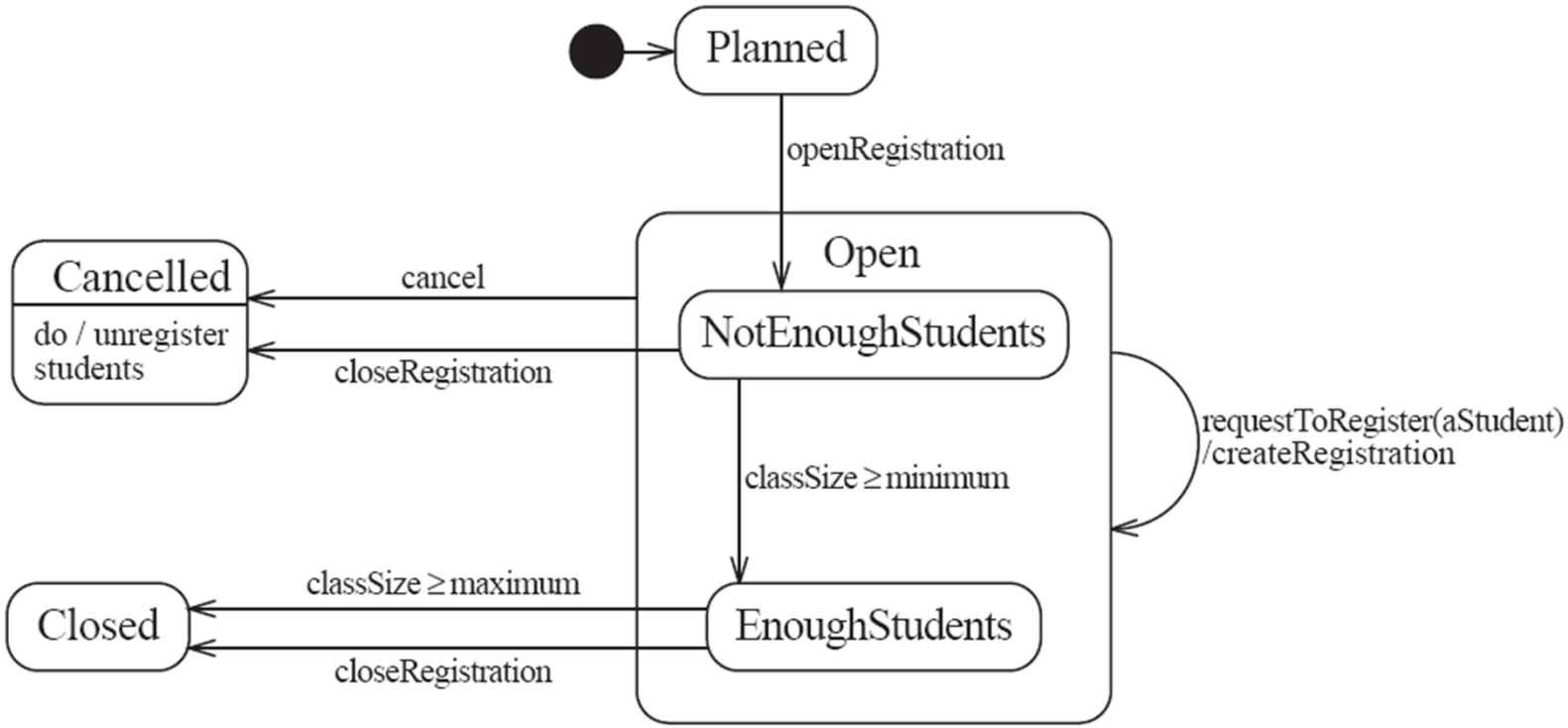


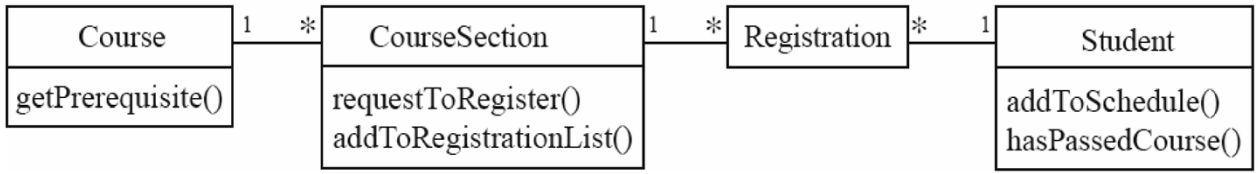
Nested substates and guard conditions

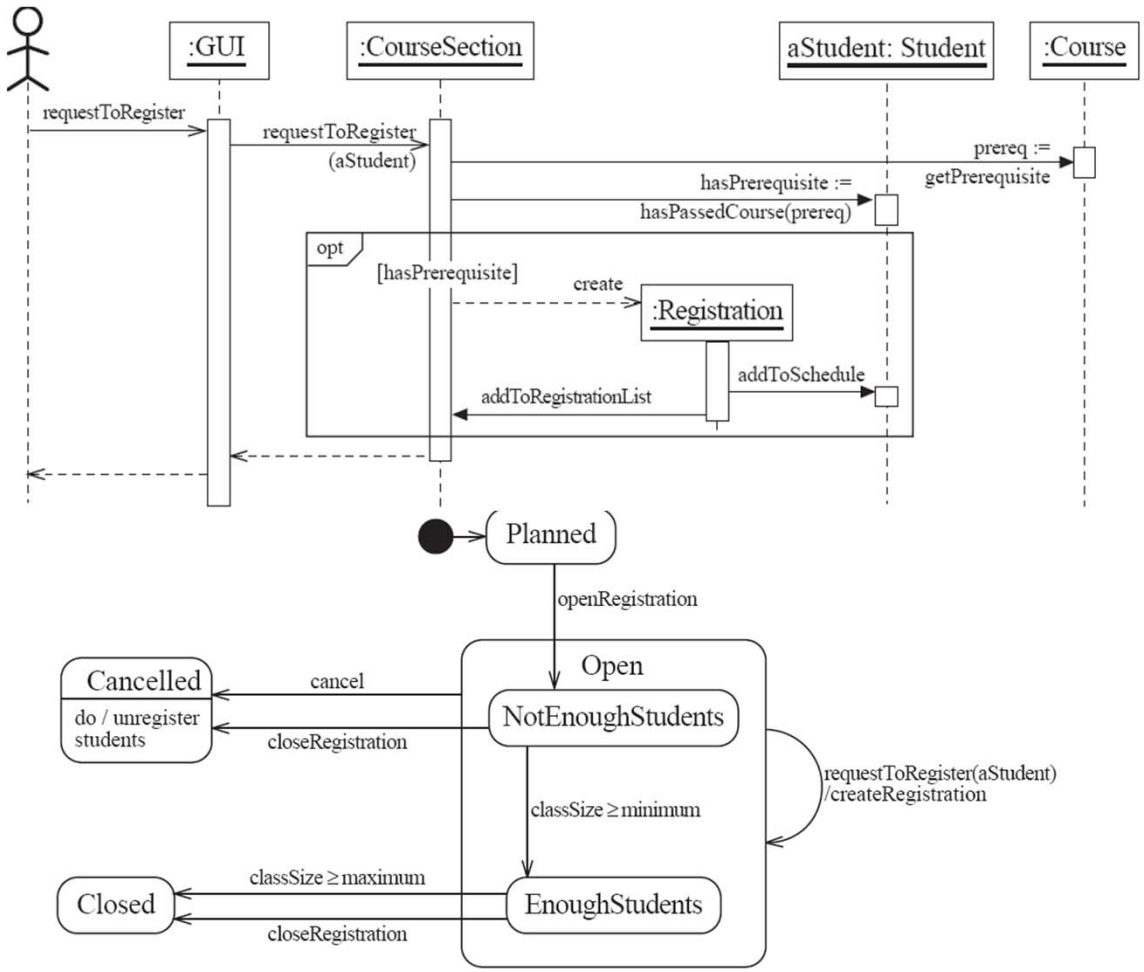
- A state diagram can be nested inside a state.
  - The states of the inner diagram are called *substates*.



State diagram – an example with substates




Course
getPrerequisite()
1
*
CourseSection
requestToRegister()
addToRegistrationList()
1
*
Registration
*
1
Student
addToSchedule()
hasPassedCourse()


:GUI
:CourseSection
aStudent: Student
:Course
requestToRegister
requestToRegister
(aStudent)
prereq :=
getPrerequisite
hasPrerequisite :=
hasPassedCourse(prereq)
opt
[hasPrerequisite]
create
:Registration
addToSchedule
addToRegistrationList
Planned
openRegistration
Open
Cancelled
do / unregister
students
cancel
closeRegistration
NotEnoughStudents
requestToRegister(aStudent)
/createRegistration
classSize ≥ minimum
classSize ≥ maximum
Closed
EnoughStudents
closeRegistration

Example

Difficulties and Risks in Modelling Interactions and Behaviour

- Dynamic modelling is a difficult skill
  - In a large system there are a very large number of possible paths a system can take.
  - It is hard to choose the classes to which to allocate each behaviour:
    - *Ensure that skilled developers lead the process, and ensure that all aspects of your models are properly reviewed.*
    - *Work iteratively:*
      - *Develop initial class diagrams, use cases, responsibilities, interaction diagrams and state diagrams;*
      - *Then go back and verify that all of these are consistent, modifying them as necessary.*
    - *Drawing different diagrams that capture related, but distinct, information will often highlight problems.*