Dynamic Modelling
UML Sequence Diagrams

CITS4401 Software Requirements and Design

Week 6

Recap

1. UML Class diagrams (what they are for and how to read them)
2. Discovering objects (noun discovery method)
3. Discovering associations (Class, Responsibilities, Collaboration (CRC) method)

UML class diagrams

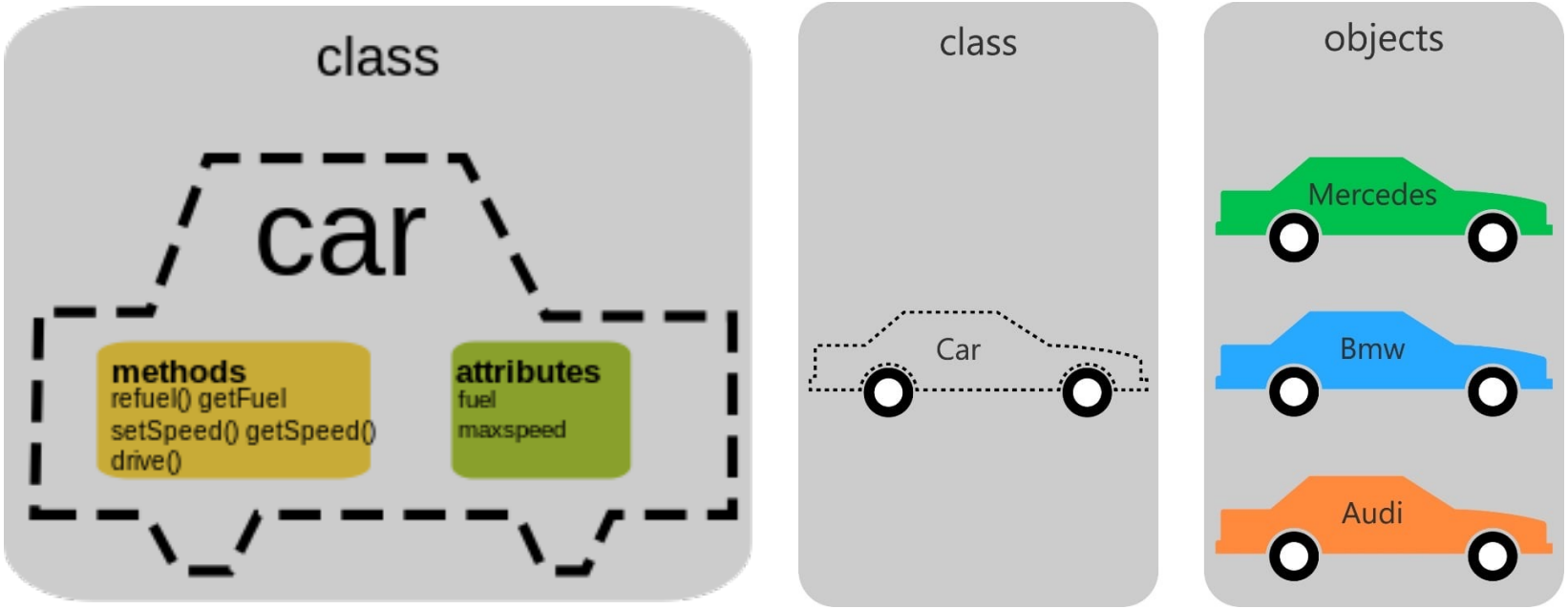
UML Class Diagrams describe the **static** structure of the system

- classes,
- class attributes,
- associations between classes,
- association roles and multiplicity

classes: *features and facts about the problem domain which matter in the system we are building to support it*

Reference: UML Distilled by Martin Fowler, Chapter 3

OOP concept


class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed
class
Car
objects
Mercedes
Bmw
Audi

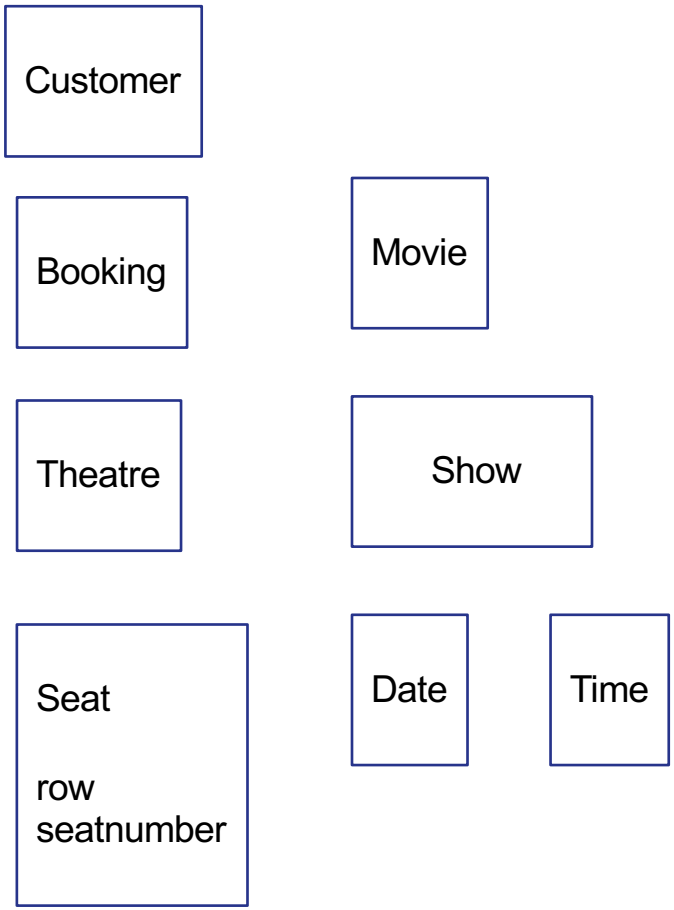
Actors, Objects and Classes

- What is the difference between an actor and a class and an object?
- Actor:
  - An entity outside the system to be modeled, interacting with the system (“Driver”)
- Class:
  - An abstraction modeling of an entity in the problem domain, inside the system to be modeled (“Car”)
- Object:
  - A specific instance of a class (“my white Toyota Corolla hybrid”).

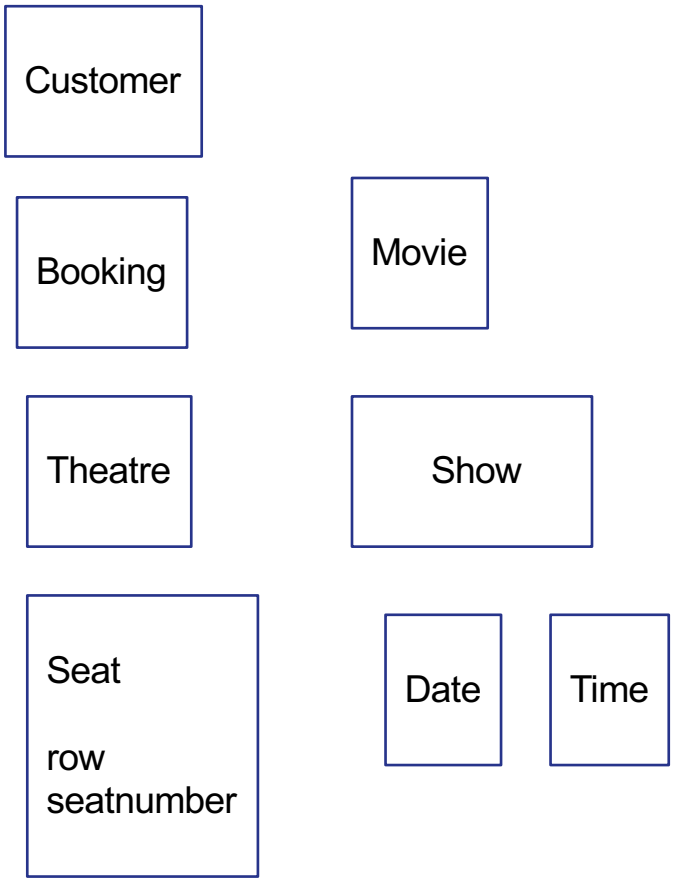
Case Study: Cinema booking system

- The cinema booking system should store seat bookings for multiple theatres. Each theatre has seats arranged in rows.
- Customers can reserve seats and are given a row number and a seat number. They may request bookings of several adjoining seats.
- Each booking is for a particular show (that is a screening of a given at a certain time).
- Shows are at an assigned date and time and are scheduled for a theatre where they are screened.
- The system stores the customer's telephone number.
- Reference Barnes and Kolling Objects First Chapter 15
- Video Notes: VN 13.1 Using the noun-verb method for application design

Cinema Booking System
Nouns = potential classes



Cinema Booking System



Verbs = potential associations

- stores (seat booking)
- has (seats)
- reserves (seats)
- Is given (row and seat number)
- Requests (seat booking)
- is scheduled (in theatre)

Class-Responsibility-Collaborator Model

- the class name of an object
  - creates a vocabulary for discussing a design
- responsibilities of an object
  - identify problems to be solved
  - a handful of short verb phrases, each containing an active verb
- collaborators of an object are
  - other objects which will send or be sent messages in the context of satisfying responsibilities
- CRC cards are a brainstorming tool for OO design
- Recommended for Extreme Programming XP methodology

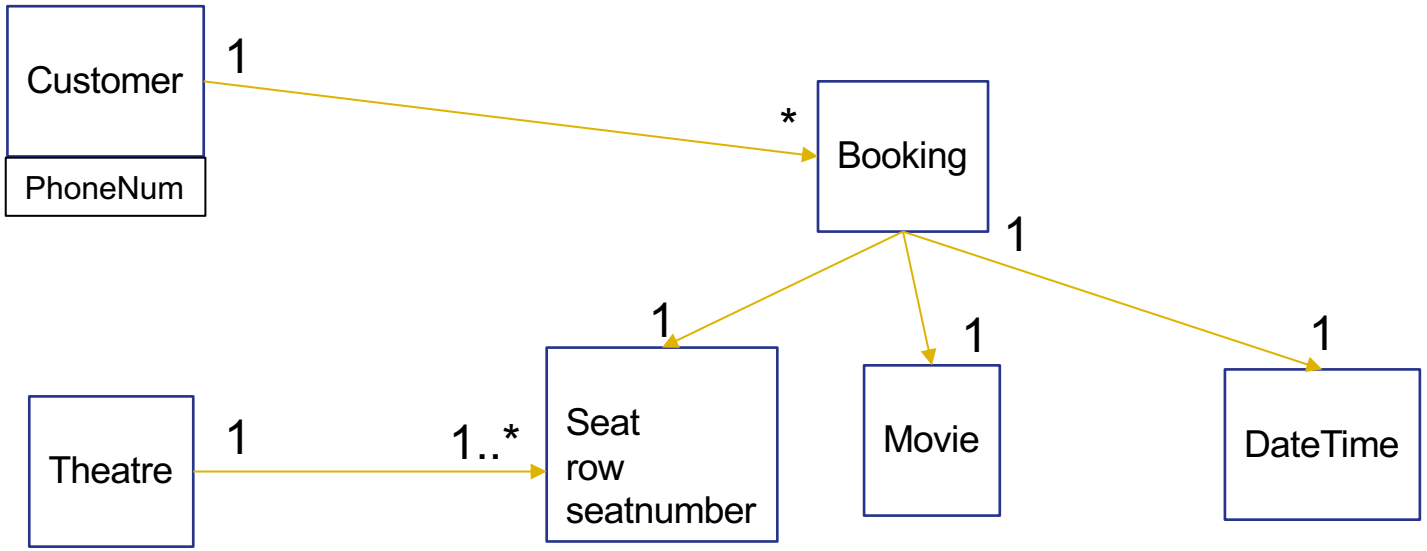
Cinema Booking Scenario CRC

- Alice selects the movie Star Wars at the Apollo Theatre
- She requests the 8pm showing on 24 March 2020
- The booking system sends the request to Apollo
- Alice is allocated seat R18

Customer	
Selects theatre, movie and show date and time	Show
Receives tickets	Theatre

Theatre	
Receives booking request	Customer, Show
Allocates tickets	Customer, Show

Cinema Booking System
UML class diagram (version 0)



BookingSystem
(could store a collection of shows to book)

This is a model in progress
Ideas and changes welcome

Lecture Overview

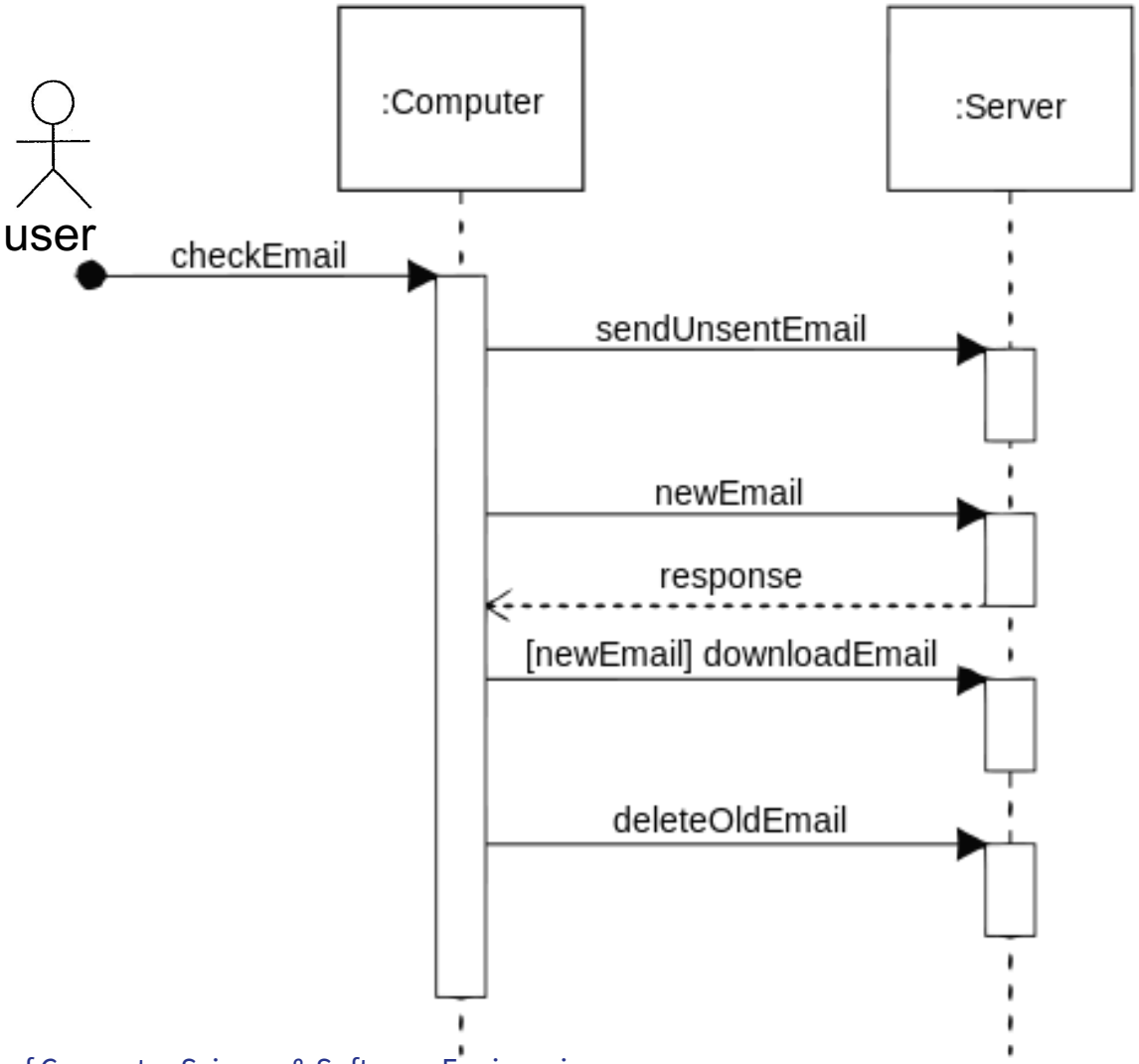
- What does a UML sequence diagram look like?
- What are the components of a sequence diagram?
- How do you create a sequence diagram?
- What about more complex situations?
- When to use a sequence diagram (and when not)

Main reference for this lecture: Martin Fowler, UML Distilled, Chapter 4

Interation in UML

- **Interaction diagrams** describe how groups of objects collaborate in some behaviour.
- The UML defines several forms of interaction diagram, of which the most common is the **sequence diagram**.
- Typically, a sequence diagram captures the behaviour of a single scenario.
- The diagram shows a number of example objects and the messages that are passed between these objects within the use case.

UML sequence diagram

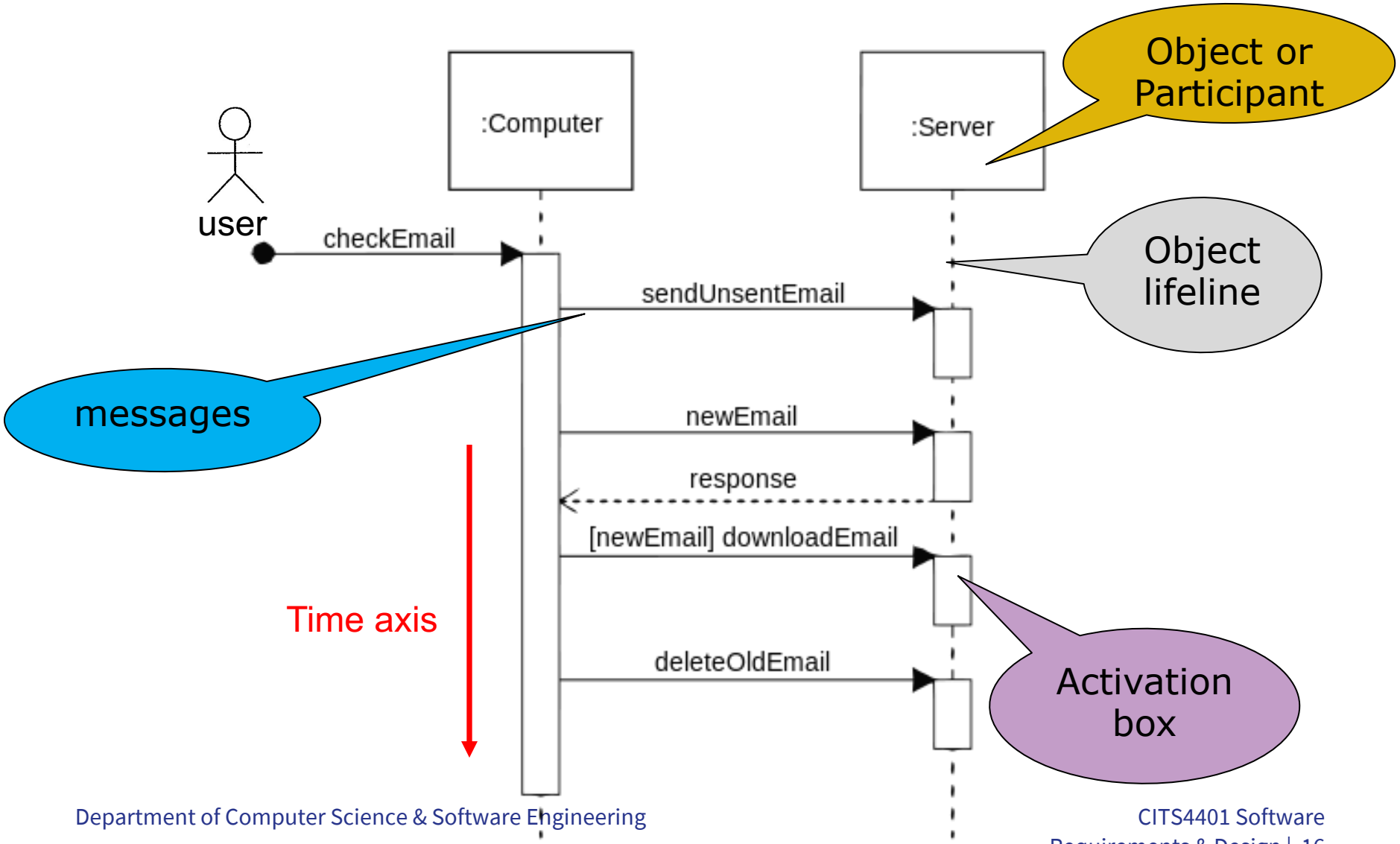


Sequence Diagrams

- **Sequence diagrams** represent
  the interaction of **participants** (horizontally)
  over **time** (vertically)
  using **messages** to communicate

- Most of the time, you can think of the participants in an interaction diagram as objects (see Fowler for discussion)

Example: user checks email



Sequence Diagrams

- **Sequence diagrams** represent the interaction of objects (horizontally) over time (vertically).
- SD ties use cases with objects, showing how the **behaviour of a use case is distributed** amongst its participating objects
- SD provide a shift in perspective, allowing developers to **find missing objects** or resolve uncertainties
- Developers should focus on problematic or under-specified functionality first, since drawing Sequence Diagrams can be time consuming

Objects + Flow

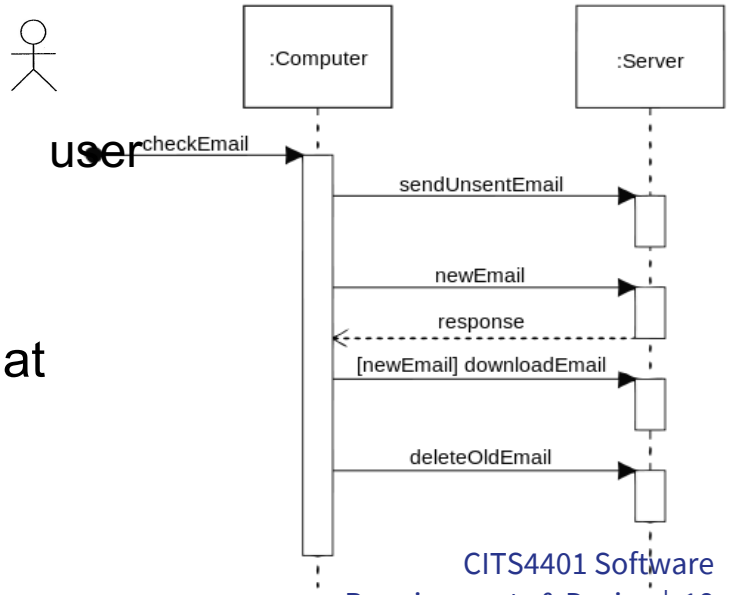
The most important components in a sequence diagrams are:

1. The **participating objects** – what are the objects associated with the use case in consideration

2. The **flow of events** between these objects – what are the messages that might be sent from the sender object to the receiver objects?

Most of the main objects should have already been identified when we come to analyze sequence diagrams.

Through dynamic modelling, new objects that were not identified may emerge.



Get the Flow of Events from a Scenario

- Flow of events from “Dial a Number” Use case:
  - Caller lifts receiver
  - Dial tone begins
  - Caller dials
  - Callee phone rings
  - Callee answers phone
  - Ringing stops
  - ....

- Heuristic:
  - An event always has a sender and a receiver. Find them for each event. These are the objects participating in the use case.

Conventions for Sequence Diagrams

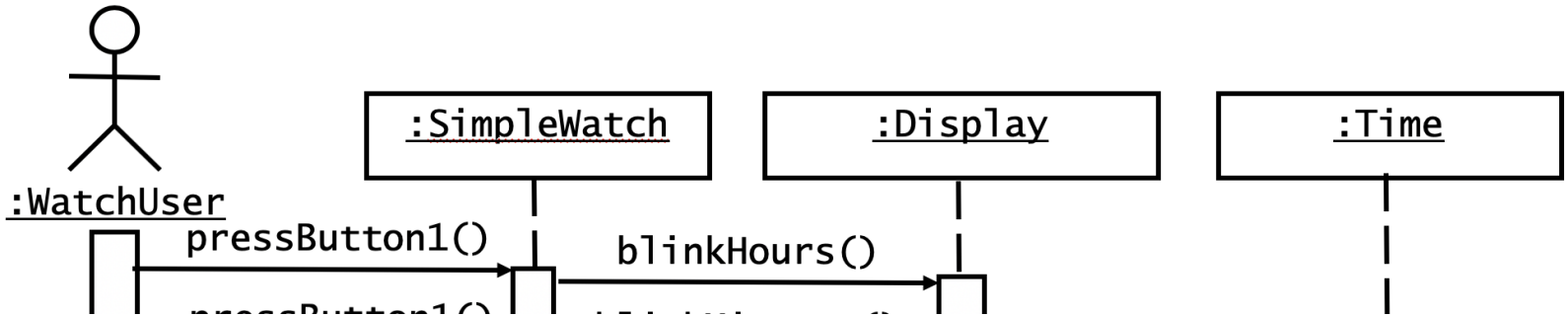
- Layout:

  1st column: corresponds to the **actor** who initiated the use case

  2nd column: Should be a **boundary object** with which the actor interacts to initiate the case

  3rd column: Should be the **control object** that manages the rest of the use case

  4th column: An **entity object** representing long-lived info tracked by the system



Conventions (2)

- Object Creation:
  - Control objects are created by boundary objects initiating the use case
  - Other boundary objects are created by control objects

- Object Access:
  - Entity objects are accessed by control and boundary objects,
  - Entity objects should never access boundary or control objects: This makes it easier to share entity objects across use cases and makes entity objects resilient against technology-induced changes in boundary objects.

Example: Order pricing scenario

- We have an order for some products made by a customer
- We are going to invoke a command to calculate its price
- To do that, the order needs to look at all the line items on the order and determine their prices
- Prices are base on the pricing rules of the order line's products
- Finally, the order needs to compute an overall discount, which is based on rules tied to the customer

Source: Fowler Chapter 4

Order Pricing Solution 1

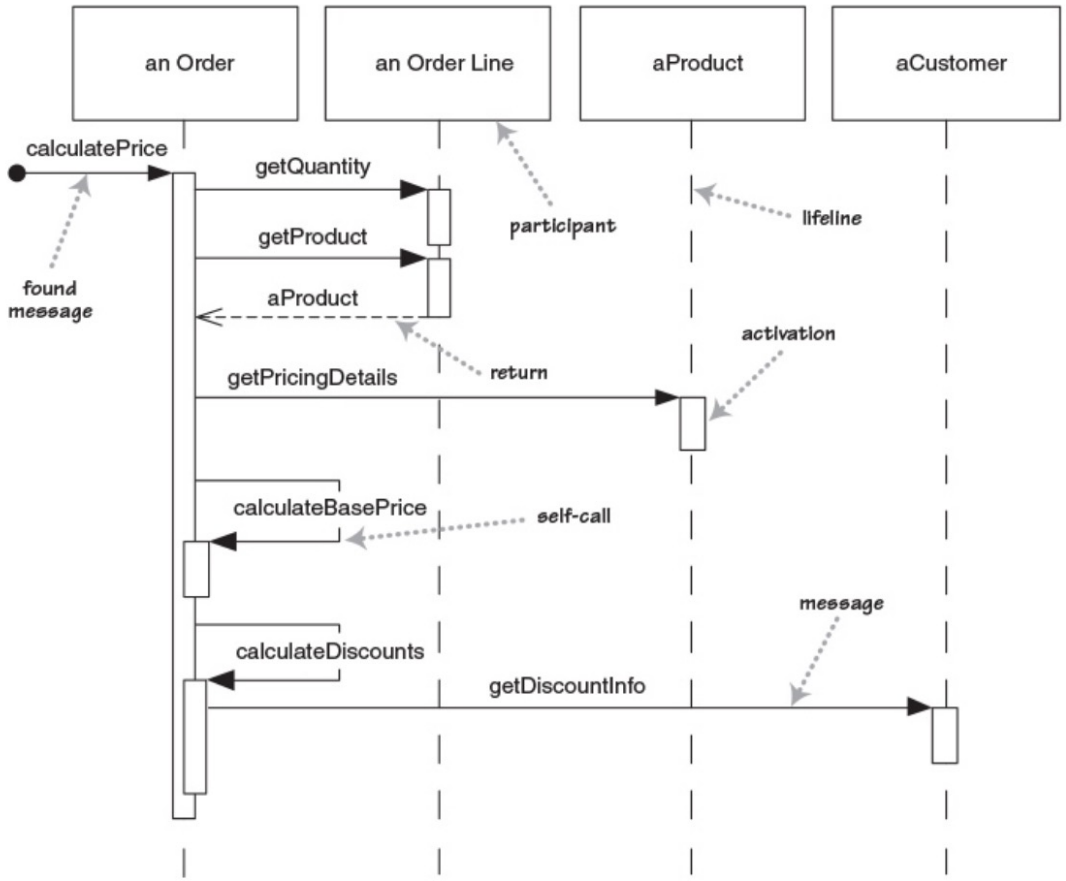


Figure 4.1 *A sequence diagram for centralized control*

SDs focus on **interactions**
SDs are **not flow charts** so don't try to capture loops or conditions

Note the return arrow is only used for the getProduct call; I did that to show the correspondence.
Some people use returns for all calls, but I prefer to use them only where they add information; otherwise, they simply clutter things

Order Pricing Solution 2

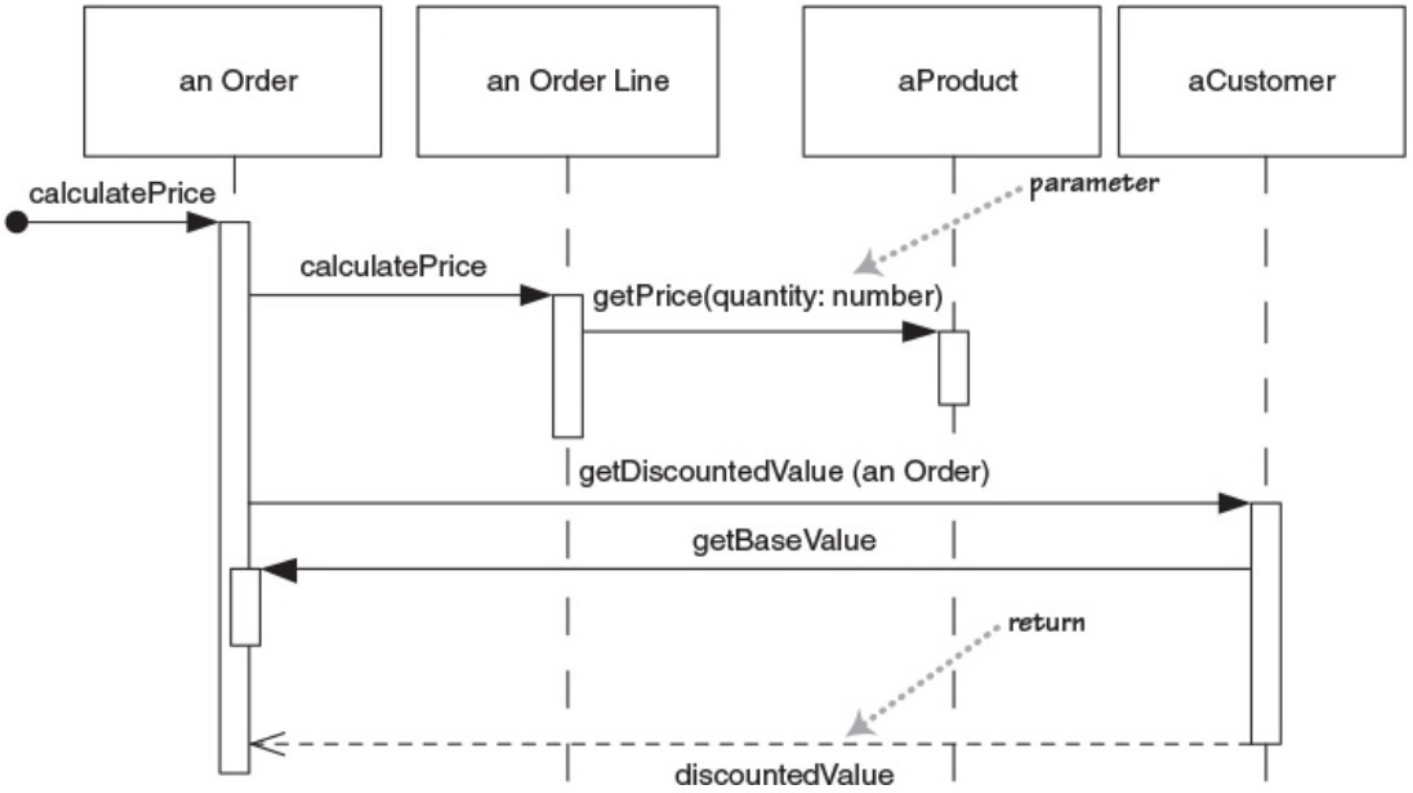


Figure 4.2 *A sequence diagram for distributed control*

Worked Example
Create new online library account

Here are the steps:

- The librarian requests the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user's details
- The user's details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account's details are then emailed to the user

Step 1: Identify participants

Here are the steps:

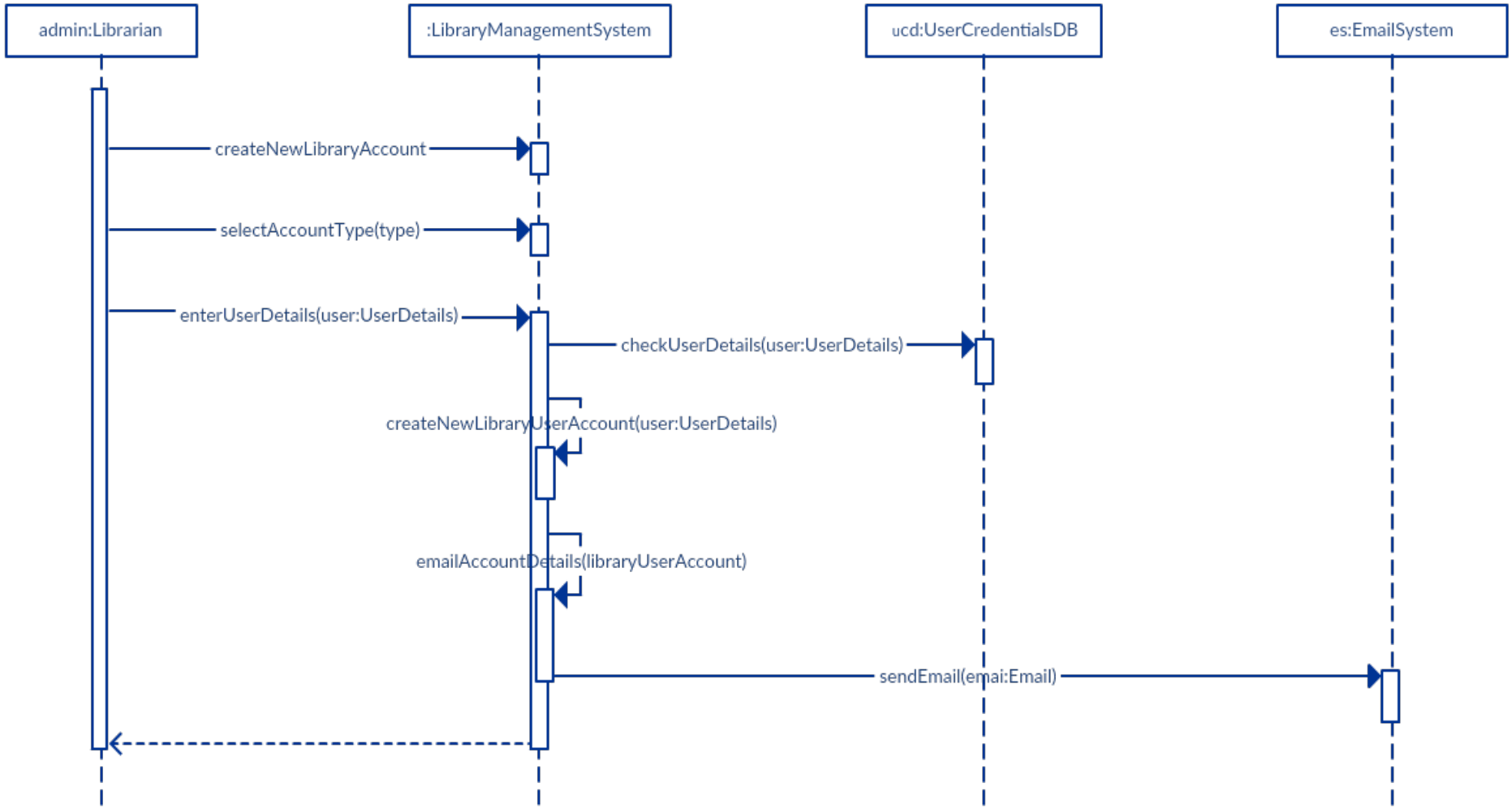
- The librarian requests the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user's details
- The user's details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account's details are then emailed to the user

Step 2: Identify events

Here are the steps:

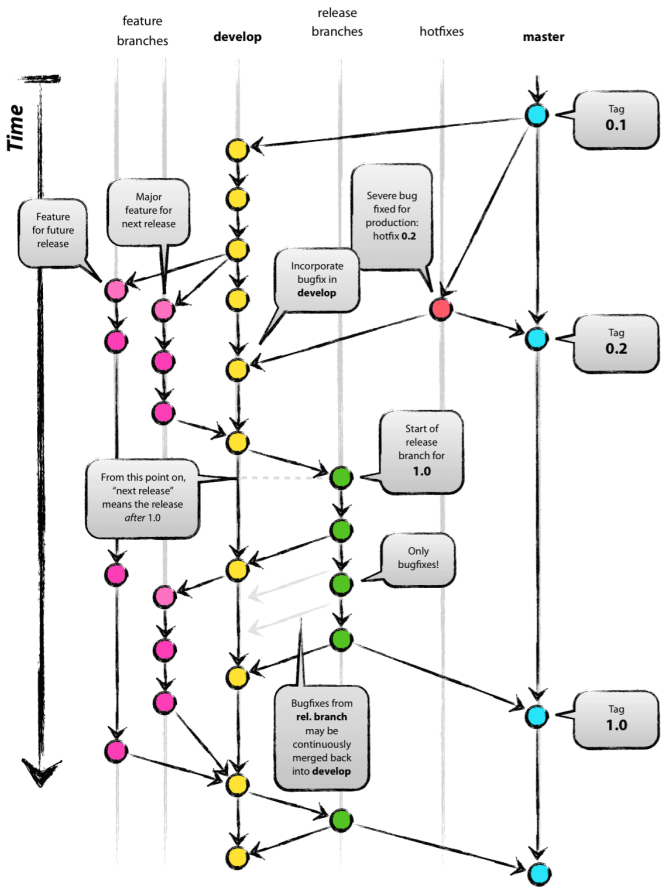
- The librarian requests the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user's details
- The user's details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account's details are then emailed to the user

Sequence diagram



More complex examples

- You don't need to make sequence diagrams for simple flows
- But for more complex situations they are good for resolving ambiguity and uncertainty.
- See https://nvie.com/posts/a-successful-git-branching-model/ for a discussion of this sequence diagram

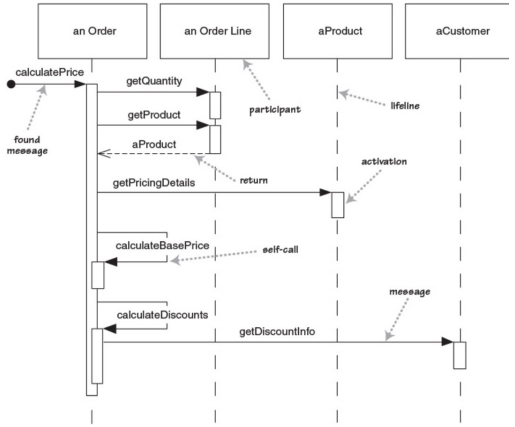


When to use Sequence Diagrams

- You should use sequence diagrams when you want to look at the behavior of several objects within a single use case.
- Sequence diagrams are good at showing collaborations among the objects;
- they are not so good at precise definition of the behavior.

- If you want to explore multiple alternative interactions quickly, you may be better off with CRC cards, as that avoids a lot of drawing and erasing. It's often handy to have a CRC card session to explore design alternatives and then use sequence diagrams to capture any interactions that you want to refer to later.

Summary



- What does a UML sequence diagram look like?
- What are the components of a sequence diagram?
  - Objects (participants) + lifelines + messages
- How do you create a sequence diagram?
  - Start with a scenario; actors to participants; identify messages; iterate!
- What about more complex situations?
  - SDs are good for resolving ambiguity and uncertainty
- When to use a sequence diagram (and when not)
  - To understand the behaviour of several objects in a single use case
  - Where collaborations between objects needs to be explored

Recommended reading

UML Distilled by Martin Fowler

Chapter 4 Sequence Diagrams

Object oriented software engineering by Bruegge & Dutoit

Section 5.4 Analysis Activities from Use Cases to Objects

Software Engineering by Pressman (different editions)

Chapter: Requirements Modelling

Section: Creating a Behavioural Model