Case Study: Restaurant Table Booking System

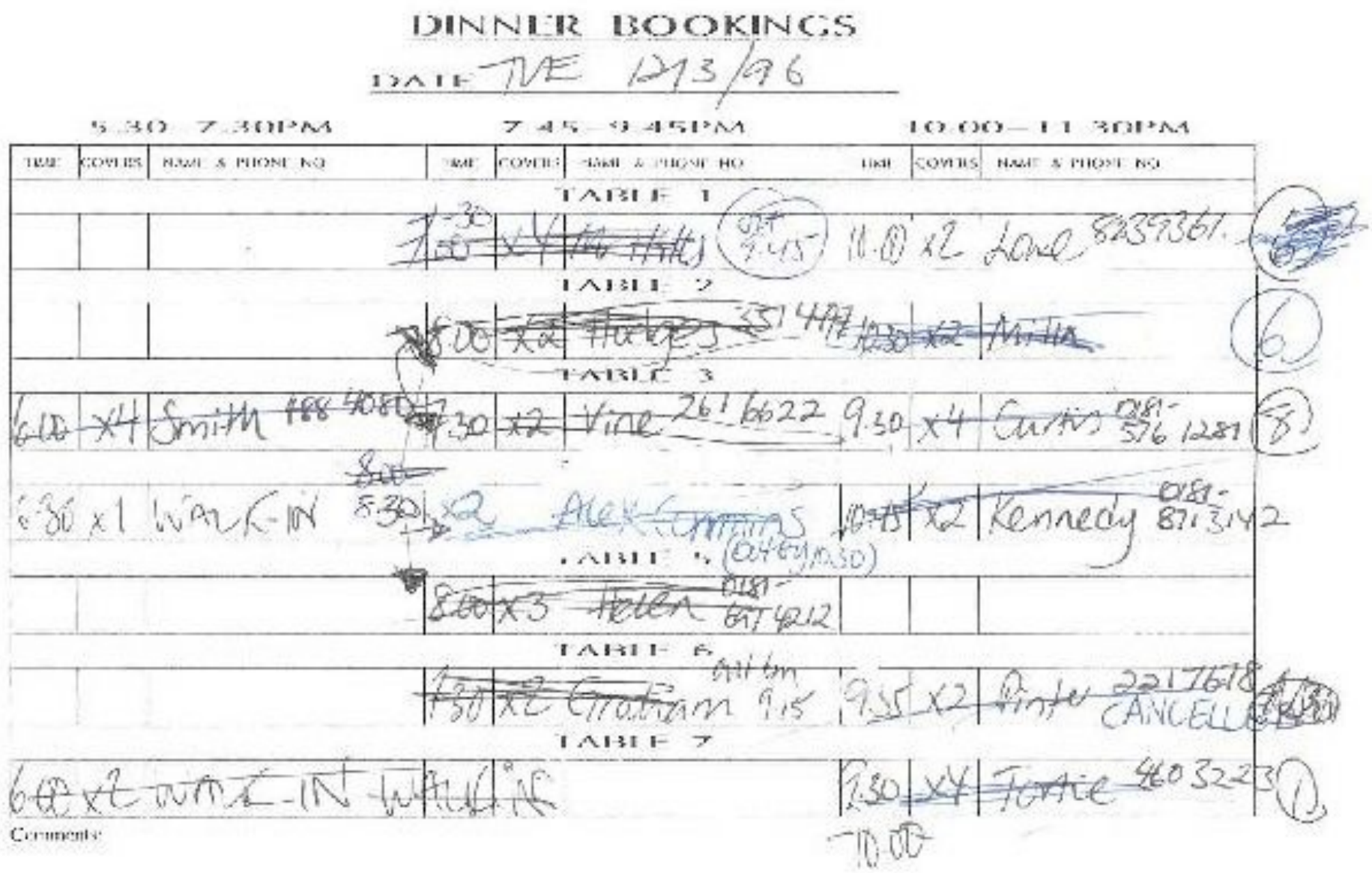
Comp 47480: Contemporary Software Development

(Slides based on Chapter 4 of *Practical Object-Oriented Design with UML* by Mark Priestly)

Case Study: Restaurant Modelling

- We introduce several UML diagrams by applying one iteration of the Unified Process to a Restaurant Table Booking system.
- Our focus will be on the analysis and design models; we won't look much at implementation and deployment.
- In this section we look at building the requirements model for the system:
  - create use case model
  - create first-cut class model (aka **domain model**)
- (Most software developers would be thinking of building a web application providing services accessed by web and mobile clients. We'll ignore that side of things and do this as a pure UML modelling exercise.)

Current System: Manual Booking Sheets



DINNER BOOKINGS

DATE TUE 12/3/96

5.30 – 7.30PM | 7.45 – 9.45PM | 10.00 – 11.30PM

TIME | COVERS | NAME & PHONE NO | TIME | COVERS | NAME & PHONE NO | TIME | COVERS | NAME & PHONE NO

TABLE 1

TABLE 2

TABLE 3

TABLE 4

TABLE 6

TABLE 7

Comments:

Current Functionality

- Advance bookings recorded on sheet
  - name and phone number of contact
  - number of diners (covers)
- 'Walk-ins' also recorded
  - customer arrives with no reservation
  - number of covers recored only
- Arrivals noted by crossing out booking
- Cancellations, table changes etc. are recorded physically on the booking sheet

What extra functionality might an automated system provide?

Note the new terminology from the restaurant domain: **walk-in** and **cover**.

Define First Iteration

- First iteration should implement a minimal useful system, based on conversation with customer
- Proposed functionality for first iteration:
  - record bookings
  - update booking sheet information (note arrivals, cancellations etc.)
- Automated system could then already replace manual sheets

Getting Started: **Use Case View**

- The **use case** view is intended to provide a structured view of the system's functionality
  - **what** the system does for the user, not **how** it does it
- Based on a description of how users interact with the system
- Supported by UML use case diagrams
- Serves as the starting point for all subsequent development
  - "use case driven development"

Use Cases

- Represent the different tasks that users can perform while interacting with the system
  - **tasks**; more than a mouseclick
- A use case isn't just "what the system does"
  - should represent some benefit for the user.
- Preliminary use case list for first iteration booking system:
  - record a new booking
  - cancel a booking
  - record the arrival of a customer
  - move a customer from one table to another
- Some important aspects are not suited for use case modelling
  - performance or timing constraints
  - UI issues
  - Internal issues, e.g. API design

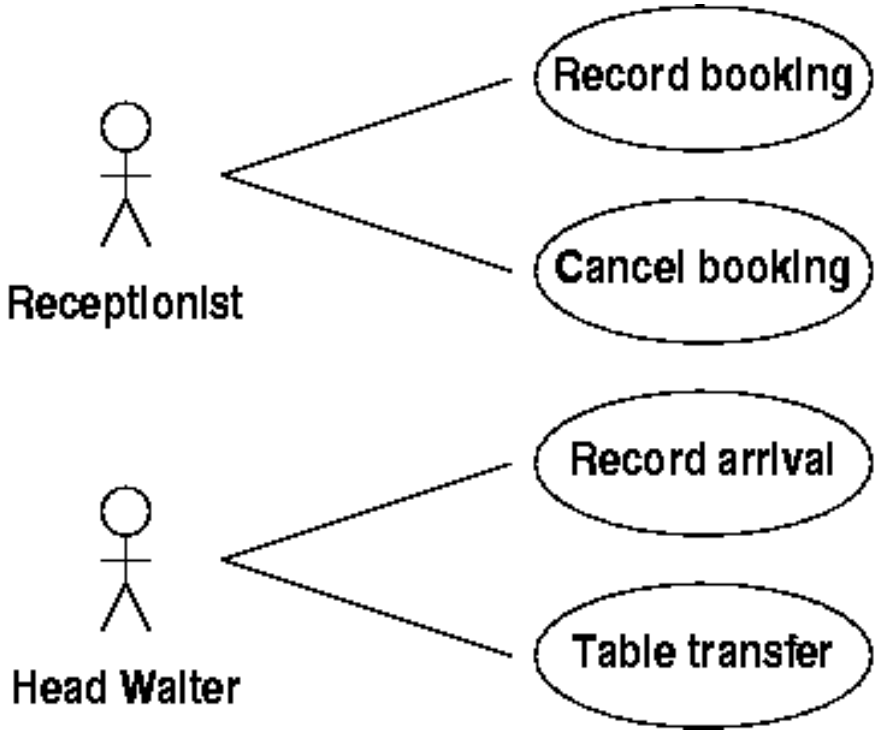
Actors

- Actors are the roles users play when interacting with a system, e.g.:
  - Receptionist (makes bookings)
  - Head waiter (assigns tables etc)
- Individual users may play one or more roles at different times
- Customers are not users of this booking system
  - => not recorded as actors
- All actors are **stakeholders**, but not all stakeholders are actors!

A **stakeholder** is anyone with an interest in the success of the system.
An **actor** interacts with the system.

Use Case Diagrams

- Shows use cases, actors and who can do what



- Simple model, you can even show it to a customer

Describing Use Cases

- A use case comprises the possible interactions that a user can have when performing a given task
- These are described as courses of events, or **scenarios**
- A full description of a use case includes:
  - a basic course of events
  - an number of **alternative** and **exceptional** courses

Basic Course of Events

- This describes what happens in the 'normal' case
- For example, for 'Record Booking':
  - Receptionist enters date
  - system displays bookings
  - Receptionist enters details
    - name, phone number, #covers.
  - system records and displays new booking
- Usually involves dialogue between system and actor

Are there reservation types not well supported by this use case?

Alternative and Exceptional Flows

Alternative Courses of Events

- Describe predicted alternative flows that the use case may take
  - Not erroneous flows
  - leads to use case goals being met

- E.g. allocate a booking to a small table
  - Receptionist enters date
  - system displays bookings
  - Receptionist enters details
  - system asks for confirmation of oversize booking
  - if "no", use case terminates with no booking made
  - if "yes", booking recorded with warning flag

Exceptional Courses of Events

- Models where the goal of the use case is not achieved
  - e.g. due to missing data or no Internet access available

- For example, if no table is available:
  - Receptionist enters date
  - system displays bookings
  - no table available: end of use case

Is this a realistic use case?

Use Case Templates

- UML does not define a standard format for use case descriptions
  - they're intended for customer communication
- Various templates have been defined to structure descriptions, including headings such as:
  - name
  - actors
  - courses of events
  - ... + many other headings

Sample Template

Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Pre-conditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Post-conditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Post-conditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Sharing Functionality between Use Cases

- Different use cases can overlap
- E.g. **Record Arrival**:
  - head waiter enters date
  - system displays bookings
  - head waiter confirms arrival for booking
  - system records this and updates display
- First two steps shared with 'Record Booking' (even though a different actor is involved)
  - We might like to model this without duplication

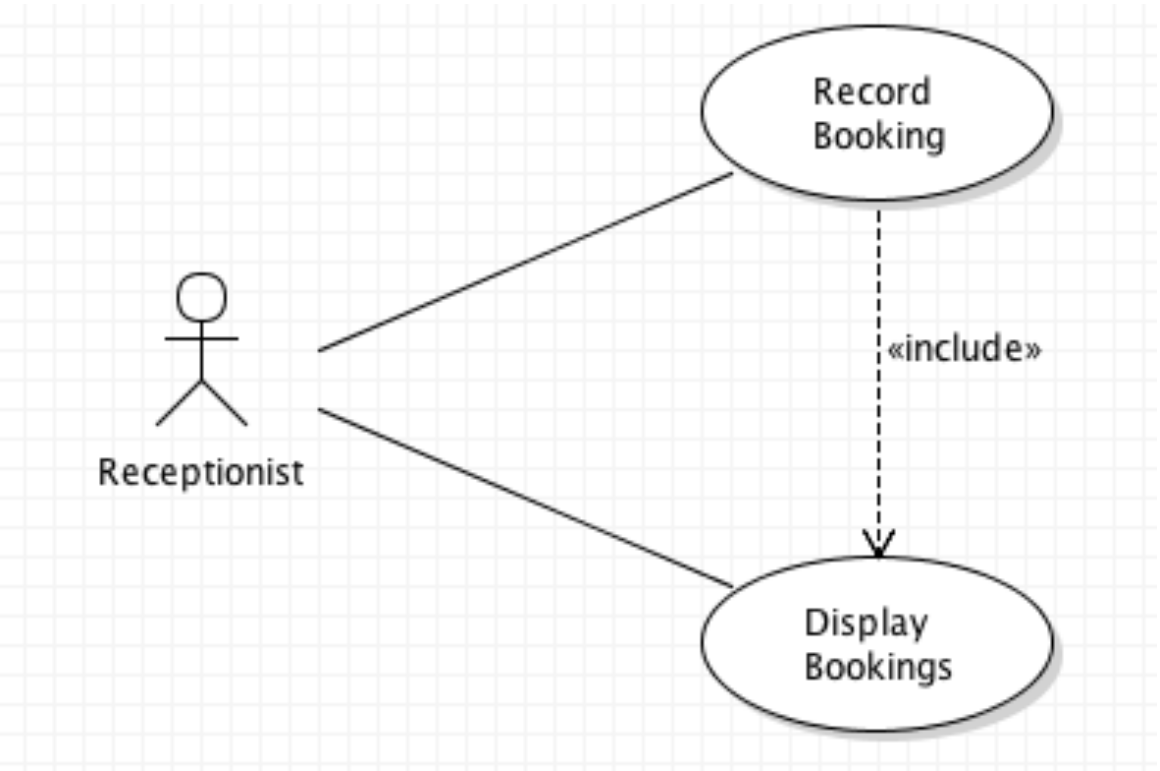
Use Case Inclusion

- Move shared functionality to a separate use case, e.g. 'Display Bookings':
  - user enters a date
  - system displays bookings for that date

- **Include** this in other use cases:
  - Receptionist performs 'Display Bookings'
  - Receptionist enters details
  - system records and displays new booking

The ‘include’ Dependency

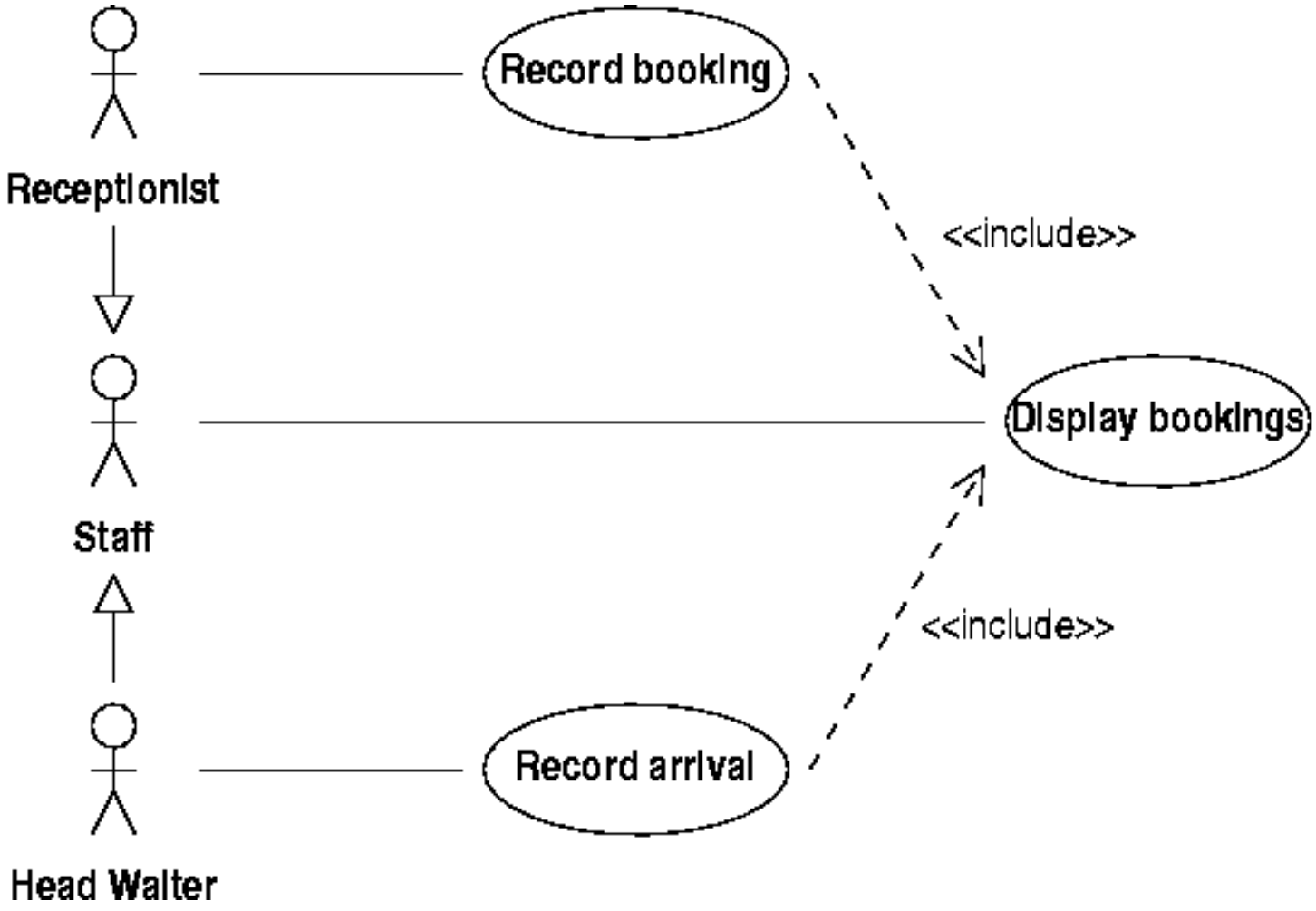
- UML shows inclusion as a **dependency** between use cases, labelled with the **stereotype** include:



Actor Generalisation

- This diagram shows that the Receptionist can display bookings without performing the including use case 'Record Booking'
  - a useful function in itself
- However, a Head Waiter can also display bookings...
  - Again, we'd like to model this without duplication
- Solution: Introduce a more general actor called Staff to model what Head Waiters and Receptionists have in common
- The initial actors are specialisations of the general actor. See next slide.

Actor Generalisation Notation

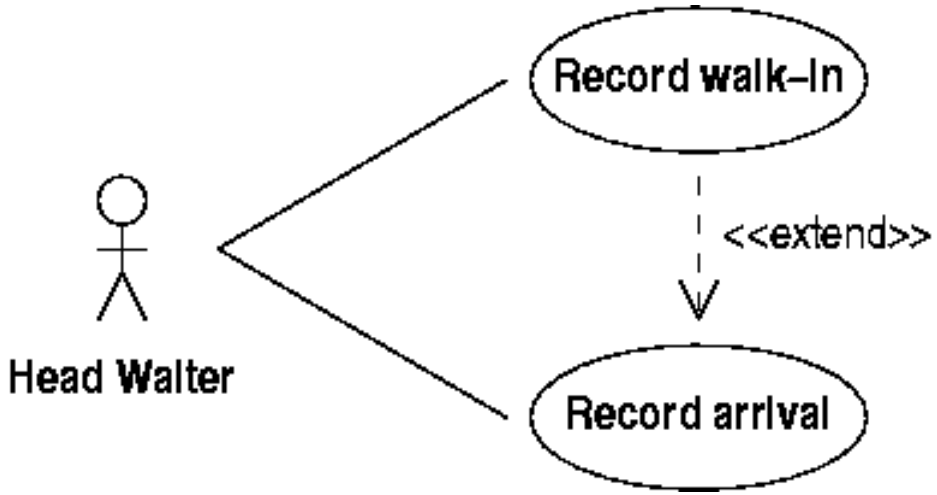


Use Case Extension

- Recording a walk-in can be modelled as a separate use case
  - a customer arrives and asks if there's a free table
- It could also be described as an exceptional source of events in the 'Record Arrival' use case.
  - Someone arrives, but there's no reservation recorded.
- Then it can be modelled as an **extension** of 'Record Arrival'
  - even without a booking, the customer stays to eat
  - See next slide.

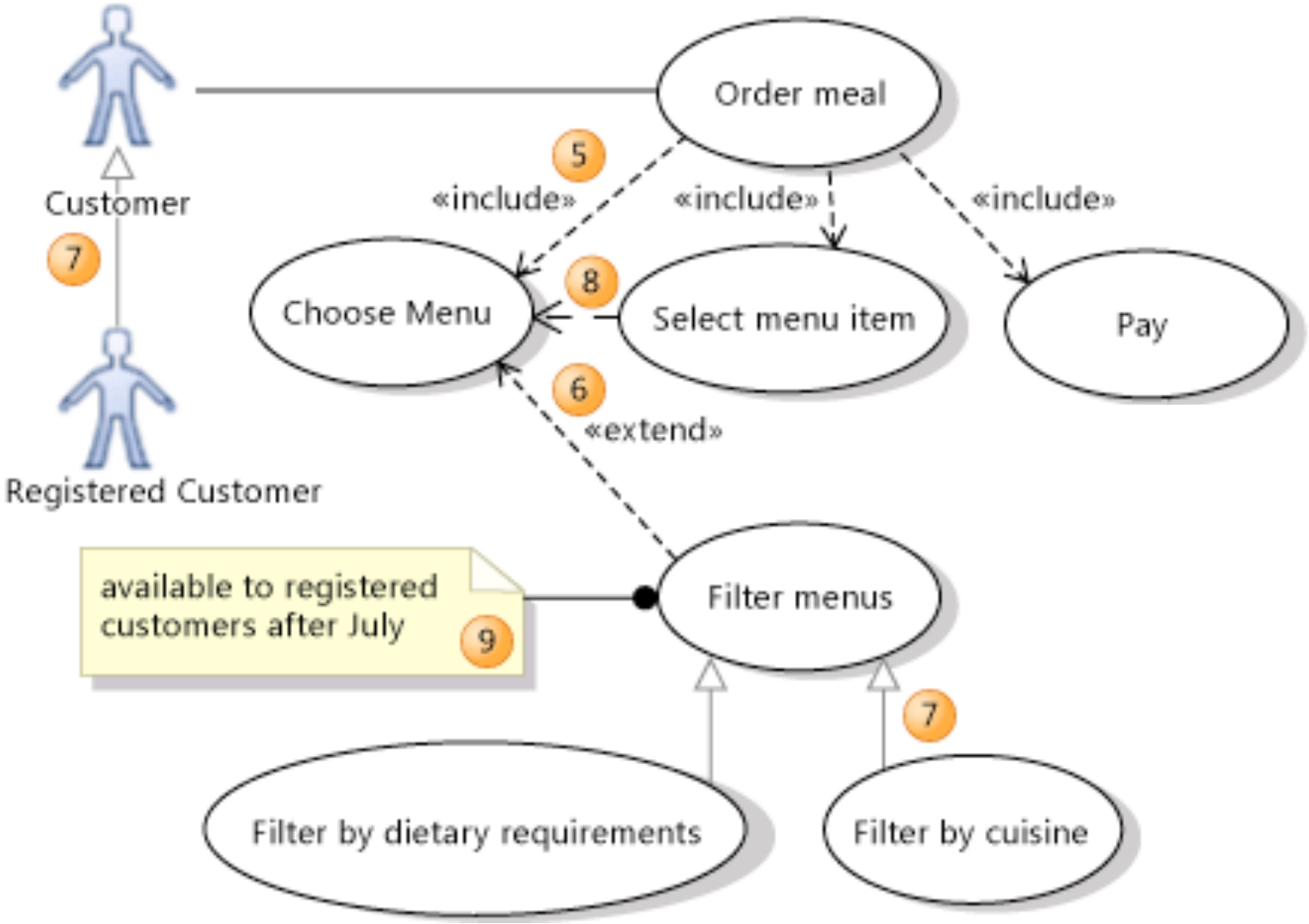
The ‘extend’ Dependency

- Use case extension is shown with a dependency thus:



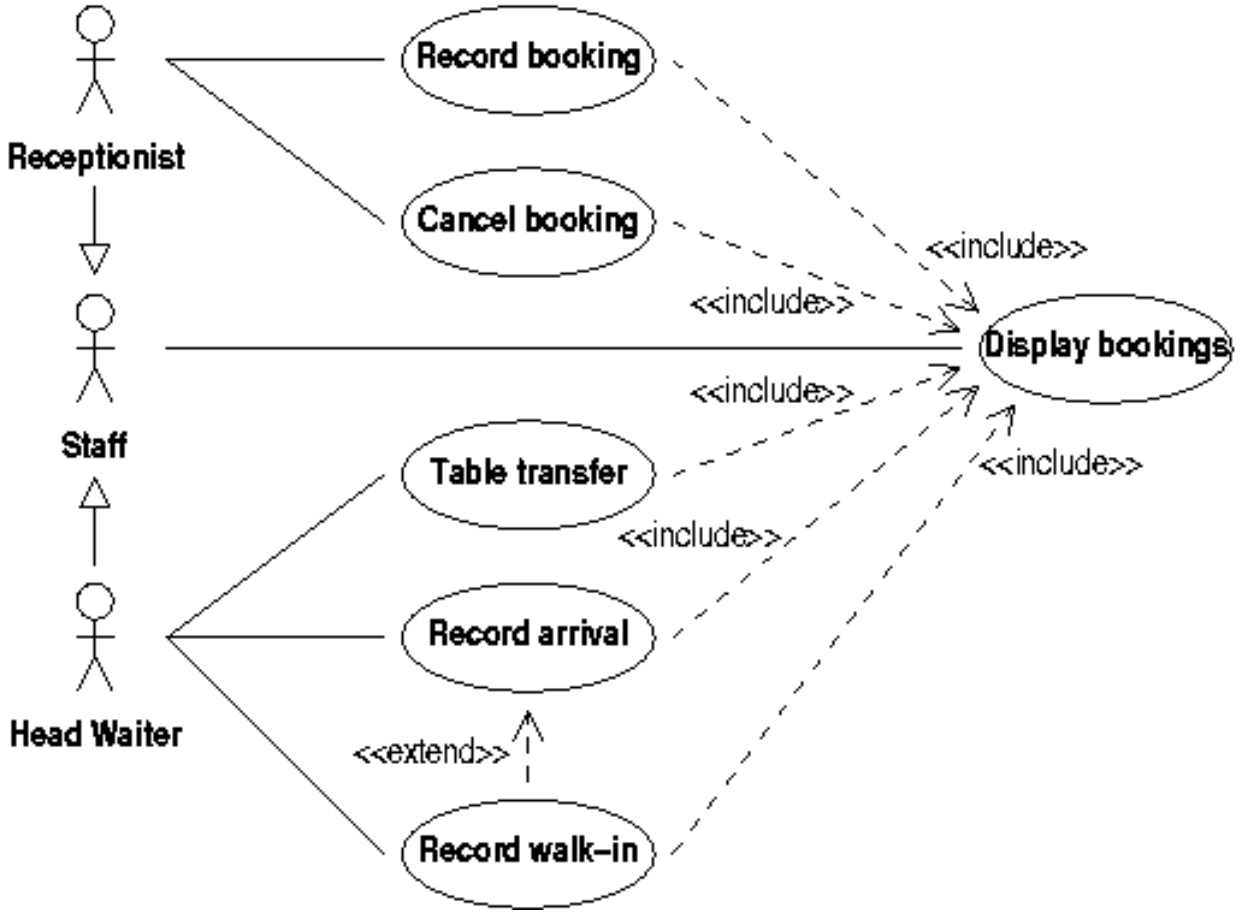
- Extensions are special cases of the base use case they extend. You might decide to implement the base case in one release, but leave the exceptional case to a later release.

Include, Extends and Generalisation



Ref: MSDN

Complete Use Case Diagram



- The model has become more complex — could you show this to a customer?

Use Case Modelling Caveats

- Keep the use case mode simple, so all stakeholders can read it
- Avoid over-using includes, extends and generalisation relationships.
  - these make the model harder to understand
  - of what use is the added complexity?
- Developers are inclined use these relationships to build **software design** into the use case model
  - this is a really bad idea, use cases are solely about what the system does, not how it does it
  - leave software design decisions to a later stage
  - don't try to identify software reuse possibilities in the use case model

Use Case Summary

- Use case describe the functionality of the system, i.e. **what** it does.
- It's a very simple model, one that customers can easily read. It is built on the basis of discussions with the customer.
- It can be made more sophisticated by using *include* and *extends* relationships
  - However, this makes the model harder for customers to read
- Use Cases are core in UML and are often used to drive the subsequent software development, termed *use case driven development*.