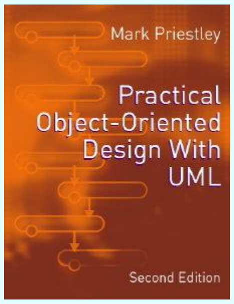
PRACTICAL OBJECT-ORIENTED DESIGN WITH UML 2e



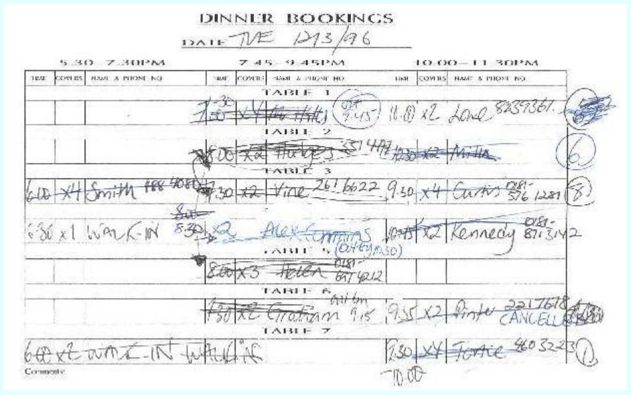
Chapter 4:
Restaurant System: Business Modelling

Business Modelling

- Early phase of development
- Inputs:
  - informal specification
- Activities:
  - create use case model
  - define use cases
  - create domain model
  - create glossary

Restaurant System

- Current system uses 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						
			7-30 ~~7.30~~	~~x4~~	~~Mr Hill~~ 0181 9.45	11.00	x2	Lowe 8239361	[illegible]
			TABLE 2						
			~~8.00~~	~~x2~~	~~Hughes 351 4497~~	~~10.30~~	~~x2~~	~~Mills~~	6
			TABLE 3						
6.00	x4	Smith ~~888 4080~~	7.30	~~x2~~	~~Vine~~ 261 6622	9.30	x4	Curtis 0181-576 1281	8
6.30	x1	WALK-IN	~~8.00~~ 8.30	x2	~~Alex Cummins~~	10.15	x2	Kennedy 0181-871 3142	
			TABLE 5 (0464 1030)						
			8.00	x3	~~Helen~~ 0181-671 4212				
			TABLE 6						
			~~7.30~~	~~x2~~	~~Graham~~ Arr'd 9.15	9.55	x2	~~Pinto 2217618~~ CANCELLED	
			TABLE 7						
6.00	x2	WALK-IN WALK-IN				9.30 10.00	x4	~~Forte~~ 460 3223	1

Comments

Current Functionality

- Advance bookings recorded on sheet
  - name and phone number of contact
  - number of diners: ‘covers’
- ‘Walk-ins’ also recorded
  - number of covers only
- Bookings allocated to a table
- Cancellations etc recorded physically on booking sheet

Define First Iteration

- First iteration should implement the minimal useful system
- Basic functionality:
  - record bookings
  - update booking sheet information
- System could then replace manual sheets

Use Case View

- This view is intended to provide a structured view of the system's functionality
- Based round 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 Cases

- The different tasks that users can perform while interacting with the system
- Preliminary list for booking system:

  1 record information about a new booking

  2 cancel a booking

  3 record the arrival of a customer

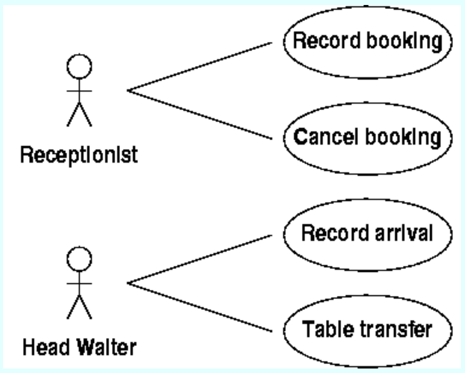
  4 move a customer from one table to another

Actors

- Actors are the roles users play when interacting with a system, eg:
  - Receptionist (makes bookings)
  - Head waiter (assigns tables etc)
- Individual users may play one or more role at different times
- Customers are not users of the system, hence not recorded as an actor

Use Case Diagrams

- Show use cases, actors and who does what



Describing Use Cases

- A use case comprises all 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’:
  1 receptionist enters date
  2 system displays bookings
  3 receptionist enters details
  4 system records and displays new booking
- Often a dialogue between system and user

Alternative Courses of Events

- Describe predicted alternative flows
- For example, if no table is available:
  1 receptionist enters date
  2 system displays bookings
  3 no table available: end of use case

Exceptional Courses of Events

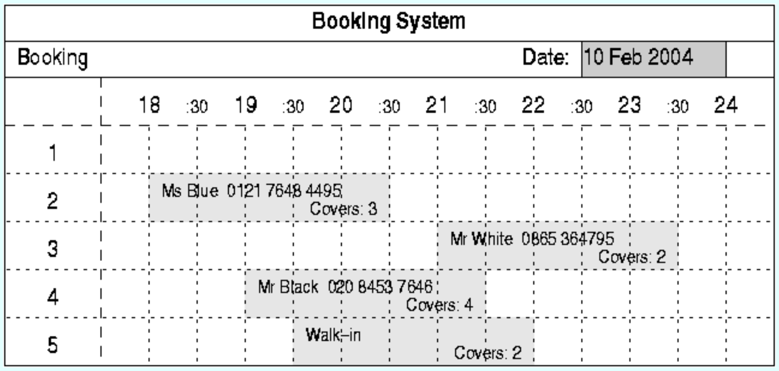
- Situations where a mistake has been made
- E.g. allocate a booking to a small table
  1 receptionist enters date
  2 system displays bookings
  3 receptionist enters details
  4 system asks for confirmation of oversize booking
  5 if “no”, use case terminates with no booking made
  6 if “yes”, booking recorded with warning flag

Use Case Templates

- UML does not define a standard format for use case descriptions
- Various *templates* have been defined to structure descriptions
- Essentially a list of subheadings such as:
  – name
  – actors
  – courses of events

User-interface Prototype

- When writing use cases, it is useful to have a rough idea of the planned user interface



Booking System

Booking Date: 10 Feb 2004

	18	:30	19	:30	20	:30	21	:30	22	:30	23	:30	24
1													
2	Ms Blue 0121 7648 4495 Covers: 3												
3							Mr White 0865 364795 Covers: 2						
4			Mr Black 020 8453 7646 Covers: 4										
5				Walk-in Covers: 2									

Shared Functionality

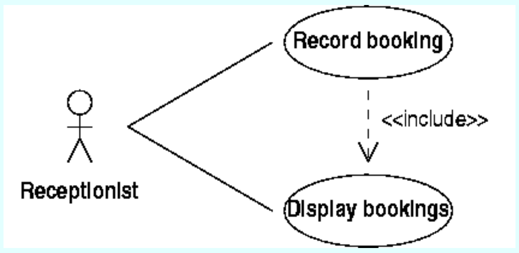
- Different use cases can overlap
- E.g. ‘Record Arrival’:
  - head waiter enters date
  - system displays bookings
  - head waiter confirms arrival for booking
  - system records this and updates display
- First two steps shared with ‘Record Booking’ (even though different actor)

Use Case Inclusion

- Move shared functionality to a separate use case, eg ‘Display Bookings’:
  - 1 user enters a date
  - 2 system displays bookings for that date
- *Include* this in other use cases:
  - 1 receptionist performs ‘Display Bookings’
  - 2 receptionist enters details
  - 3 system records and displays new booking

The ‘include’ Dependency

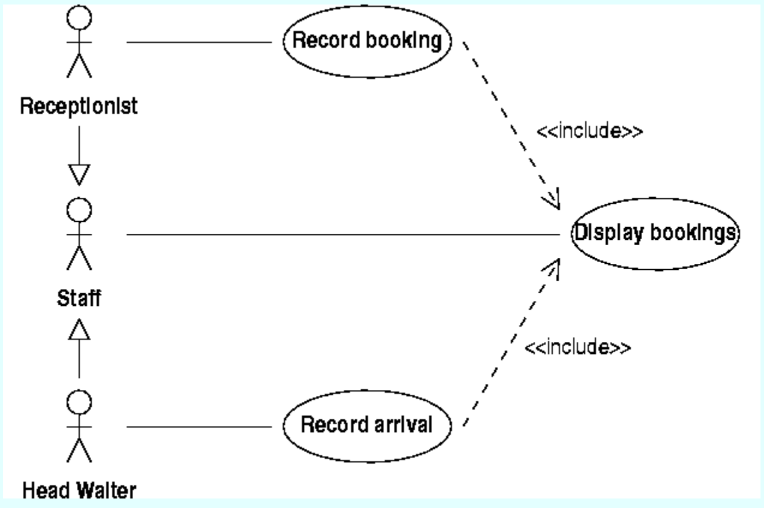
- UML shows inclusion as a *dependency* between use cases, labelled with the stereotype *include:*



Actor Generalization

- This diagram shows that the receptionist can display bookings without performing the including use case ‘Record Booking’
- Head waiters can also display bookings
- Introduce a more general actor to show what the other two actors have in common
- The initial actors are *specializations* of the general actor

Actor Generalization Notation

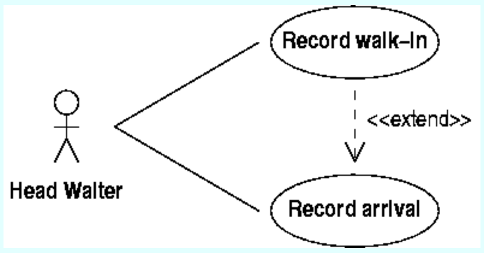


Use Case Extension

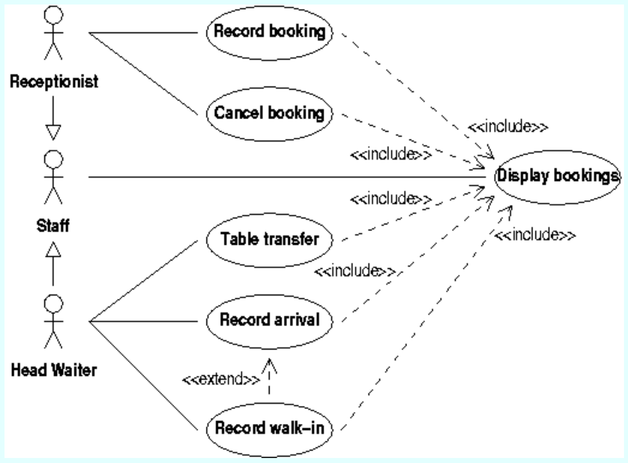
- Recording a walk-in can be described as an exceptional source of events
  - someone arrives but there's no booking recorded
- It could also be a separate use case
  - a customer arrives and asks if there's a free table
- Then it can *extend* 'Record Arrival'
  - even without a booking, the customer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

The 'extend' Dependency

- Use case extension is shown with a dependency



Complete Use Case Diagram



Domain Modelling

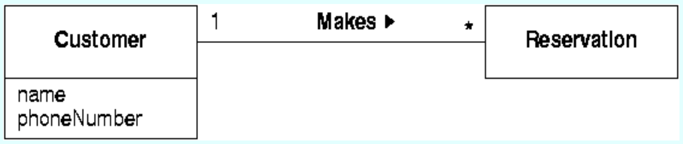
- Using UML to construct a model of the real-world system
  - similar to entity-relationship modelling
- Model recorded as a class diagram
- ‘Seamless development’
  - same notation used for analysis and design
  - design can evolve from initial domain model

Domain Model Notation

- Subset of class diagram notation
  - **classes** represent real-world entities
  - **associations** represent relationships between the entities
  - **attributes** represent the data held about entities
  - **generalization** can be used to simplify the structure of the model

Customers and Reservations

- Basic business fact: customers make reservations

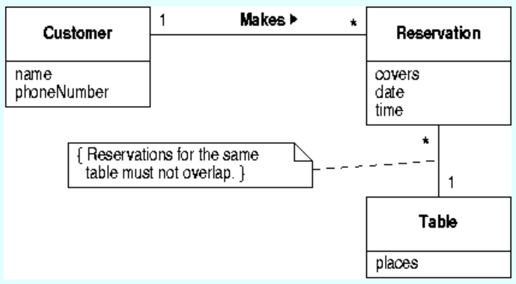


Defining a Relationship

- Give a name to the relationship
  - use a verb so that the relationship can be read as a sentence
- A customer can make many reservations
- How many people make a reservation?
  - one principal contact whose details are held
  - the expected number of diners can be modelled as an attribute of the reservation

Tables

- Is table number an attribute of ‘Reservation’?
- Better modelled as a separate class
  - tables exist even if there are no reservations
  - other attributes of tables, e.g. size, can be stored

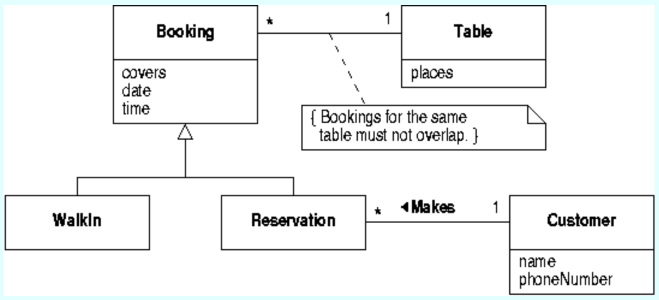


Constraints

- Not all domain properties can be shown graphically
  - e.g. it should be impossible to double-book a table
- *Constraints* add information to models
  - written in a *note* connected to the model element being constrained

Use of Generalization

- A superclass can be used to show the properties shared by different types of booking



Correctness

- How do we know when a domain model is complete?
  - we don't: there are lots of plausible models in most cases
- Domain modelling is not an end in itself, but a guide to further development
- Realizing use cases *tests* the domain model, and will usually lead to refinements

Glossaries

- Domain models capture important system concepts
- Useful to record these terms and their definitions for use throughout a project
- Do this in the form of a *glossary*

Partial Restaurant Glossary

- **Booking**: an assignment of diners to a table
- **Covers**: the number of diners for a booking
- **Customer**: a person who makes a reservation
- **Reservation**: a booking made in advance
- **Walk-in**: a booking that is not made in advance