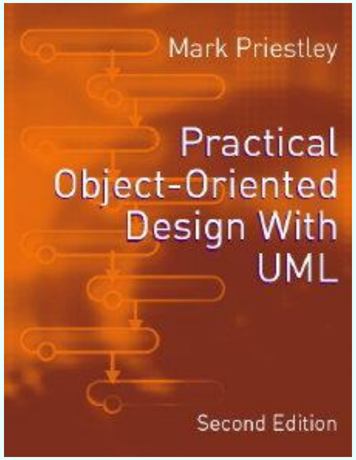
PRACTICAL OBJECT-ORIENTED DESIGN WITH UML 2e



Chapter 5: **Restaurant System: Analysis**

Mc Graw Hill Education

©The McGraw-Hill Companies, 2004

Business Modelling ขั้นที่ 1.

Analysis ขั้นที่ 2.

use case Realization. or refinement.

Analysis

- What is to be analyzed?
  - the system requirements
- Why?
  - to demonstrate their implementability
- How?
  - by drawing interaction diagrams *realizing* use cases

03.50

Analysis v. Design

- Difficult to draw a boundary
- Traditional informal distinction:
  - analysis models the real-world system
  - design models the software
- Object-oriented methods use the same notation for both activities
  - encourages ‘seamless development’ and iteration

$\searrow$ 05.25

Object Design

- We need to define attributes and operations for each class in the model
- Start from domain model, but:
  - structure of real-world application is not always the optimal structure for a software system
  - domain model does not show operations
- *Realization* identifies operations and confirms that design supports functionality

↓ 5.52.

หลักการในการวิเคราะห์

Object Responsibilities

- Each class in a system should have well-defined *responsibilities* หน้าที่หน้าที่อะไร.
  - to manage a subset of the data in the system
  - to manage some of the processing
- The responsibilities of a class should be *cohesive*
  - they should 'belong together'
  - they should form a sensible whole

$\downarrow$ 7.08

Software Architecture

- A software architecture a high level view of software, described as a number of of distinct components or subsystems together with their relationships and interaction .
- Description of UML component/deployment may be used to document architectures
- Architectures are the configurations of components that make up the systems.
- Architectural pattern is a high level pattern describing a solution at architectural level.
- Architectures are the configurations of components that make up the systems.

Software Architecture

- Data-flow: concentrates on the flow of data e.g. batch processing.
- Data-centered: focuses on centralised persistent data e.g. data base.
- Virtual-machine: layered software e.g. ISO OSI seven layer model.
- Call-return: focuses on a sequence of instruction, single thread of control.
- Independent-component: supports modifiability e.g. client server.

↓ 07.55

Software Architecture

- The UP analysis workflow includes the production of an *architectural description*
- This defines:
  - the top-level structure of subsystems
  - the role and interaction of these subsystems
- Typical architectures are codified in *patterns*
  - for example, *layered architectures*

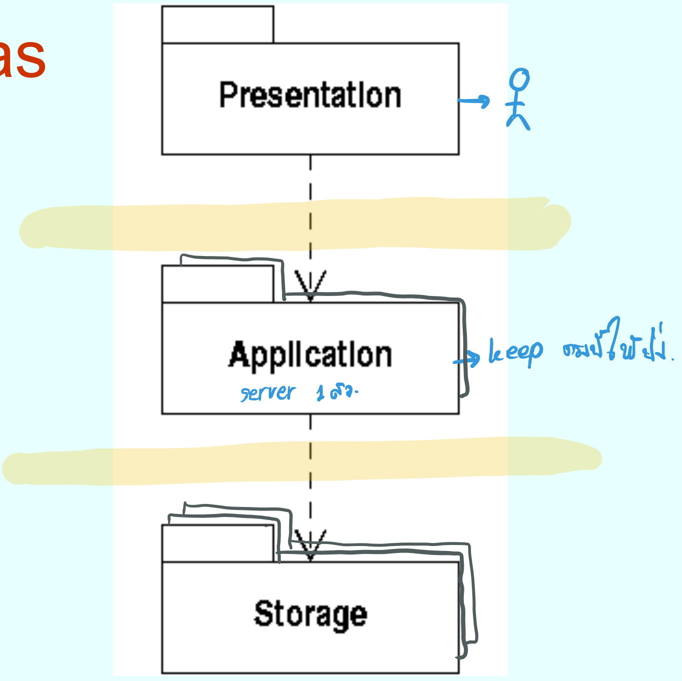
08.37

software ที่ work.

A Layered Architecture

three tiar.

- Subsystems are shown as UML *packages* linked by *dependencies*
- A dependency without a stereotype means *uses*



Mc Graw Hill Education ©The McGraw-Hill Companies, 2004

↓ 18.12

Separation of Concerns

- Layers aim to insulate a system from the effects of change
- For example, user interfaces often change
  - but the application layer does not use the presentation layer
  - so changes to system should be restricted to presentation layer classes
- Similarly, details of persistent data storage are separated from application logic

Analysis Class Stereotypes

- Within this architecture objects can have various typical roles
  - *boundary* objects interact with outside actors
  - *control* objects manage use case behaviour
  - *entity* objects maintain data
- These are represented explicitly in UML by using *analysis class stereotypes*

↓ 23.30.

Class Stereotype Notation

- Stereotypes can be text or a graphic icon
- The icon can replace the normal class box

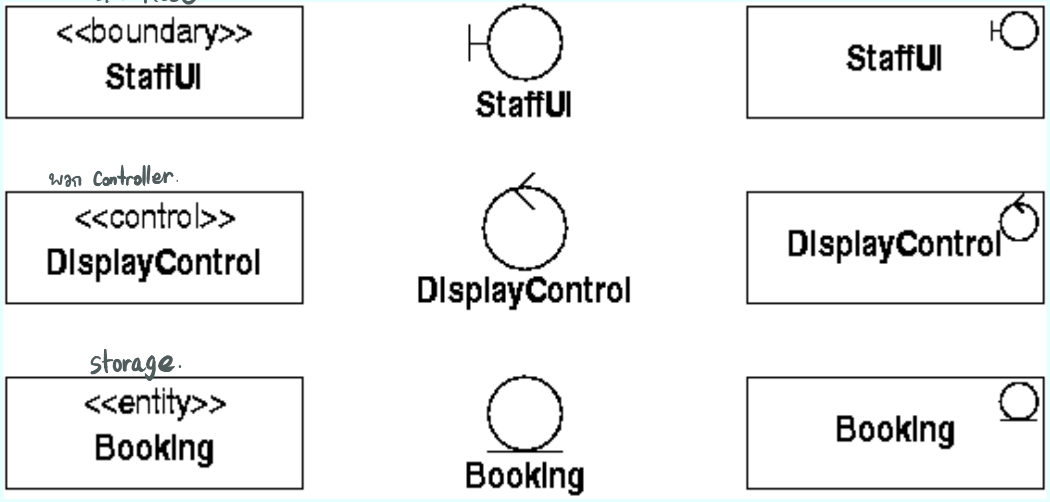
อยู่ Tier ไหน.

Text Mode

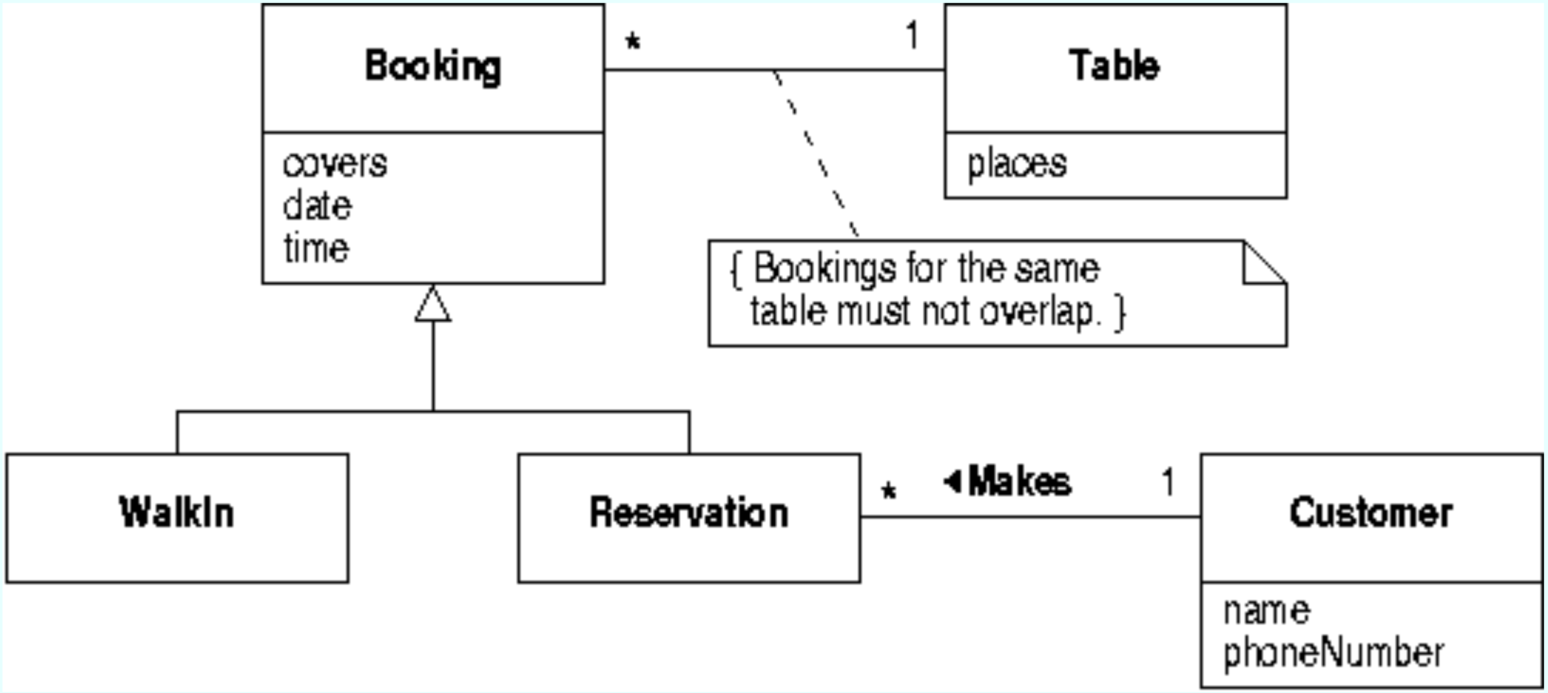
ใน EA.

พวก Controller.

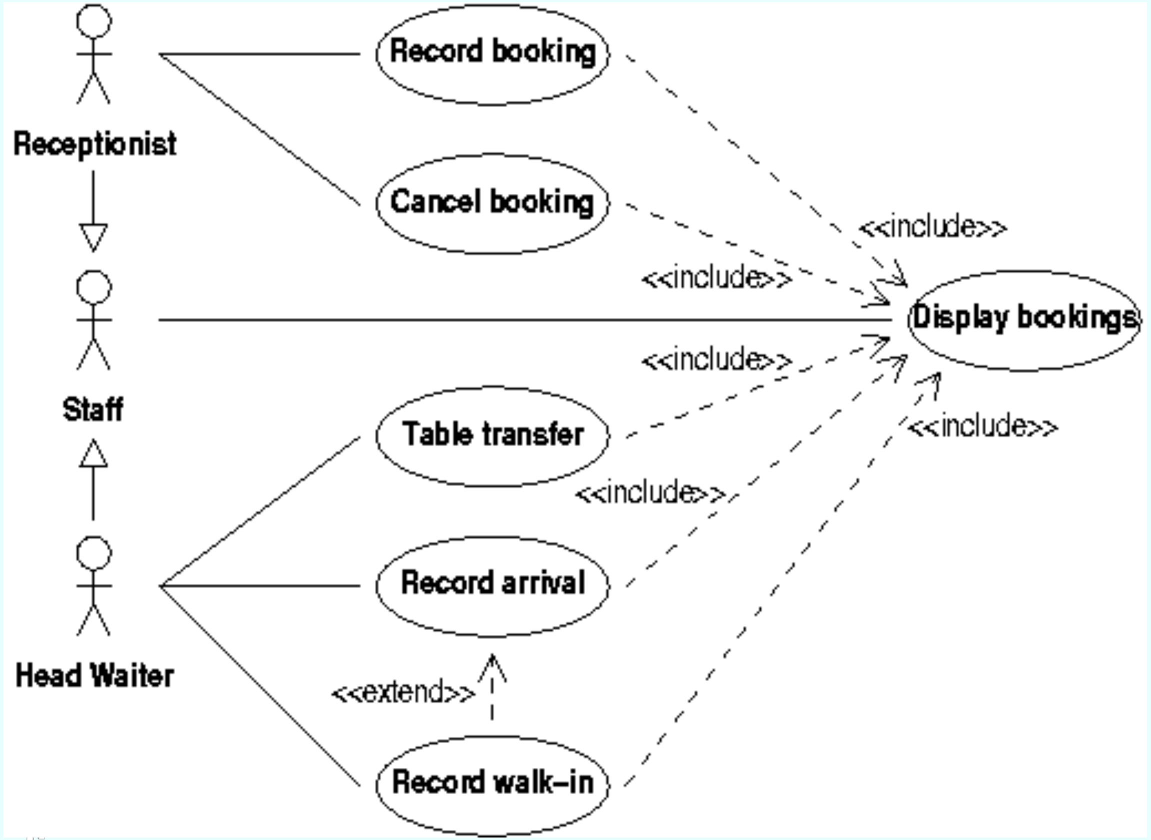
storage.



Restaurant Domain Model(4.10)



Restaurant Use Case Diagram(4.7)



Mc Graw Hill Education

©The McGraw-Hill Companies, 2004

27.30.

Display Booking

Sequence Diagram.

สลับการทำงาน

Use Case Realization

- Begin with functionality in application layer
- 'Display Bookings': simple dialogue
  - – the user provides the required date
  - – the system response is to update the display
- Initial realization consists of
  - – instance of the 'Staff' actor
  - – an object representing the system
  - – message(s) passed between them

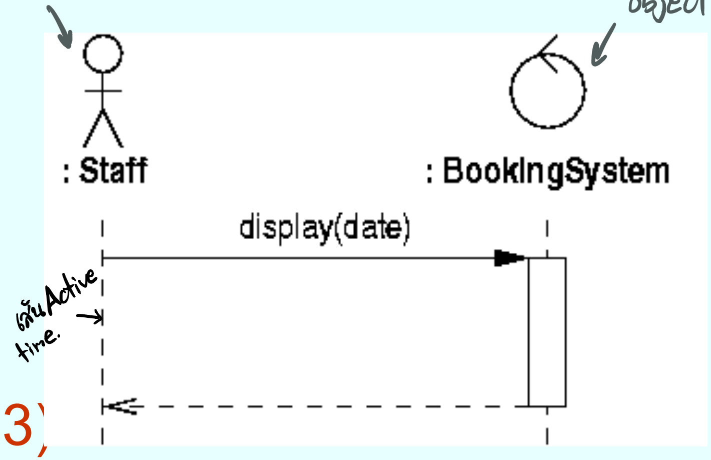
30.04

System Messages

- *System messages* are sent by an actor
- Represent system by a *controller*
  - initially analysing use case behaviour, not I/O

object. → Role. แทน Object หลายๆ ตัว.

↳ $\boxed{\text{: staff.}}$



  - (Fig. 5.3)

keyword : Object Responability.

Sequence Diagrams

- Time passes from top to bottom
- Instances of classes and actors at top
  - only show those participating in this interaction
  - each instance has a *lifeline*
- Messages shown as arrows between lifelines
  - labelled with operation name and parameters
  - return messages (dashed) show return of control
  - *activations* show when receiver has control

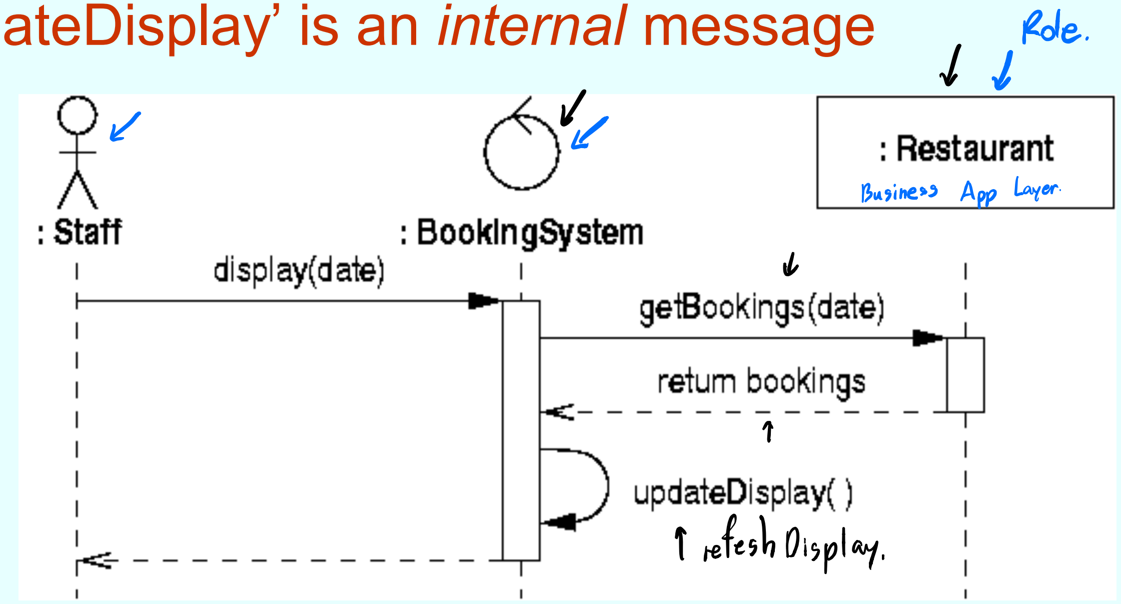
Accessing Bookings

- How does the system retrieve the bookings to display?
- Which object should have the responsibility to keep track of all bookings ?
  - if this was an additional responsibility of the 'BookingSystem' object it would lose *cohesion*
  - so define a new 'Restaurant' object with the responsibility to manage booking data

38.56.

Retrieving Bookings

- Add a message to get relevant bookings
- 'updateDisplay' is an *internal* message



- (fig. 5.4)

ขั้นตอนในการใช้เคสนี้เรียก Scenario.

↓ 40.30

Retrieving Booking Details

Restaurant.

get Booking.

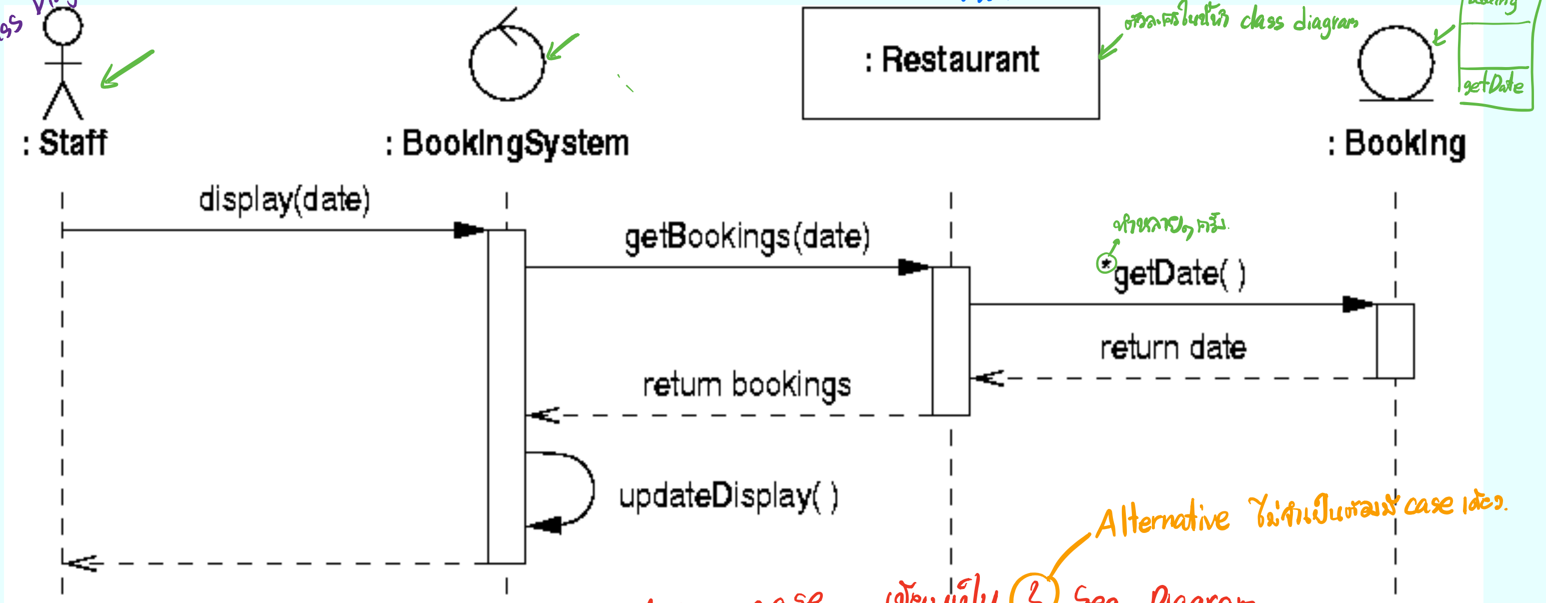
Seq Diagram

↑

เอาข้อมูลไปเขียน Class Diagram.

- Dates of individual bookings will need to be checked by the ‘Restaurant’ object (fig. 5.5)

หน้าที่ของมัน.



1 use case เขียนเป็น 3 Seq Diagram.

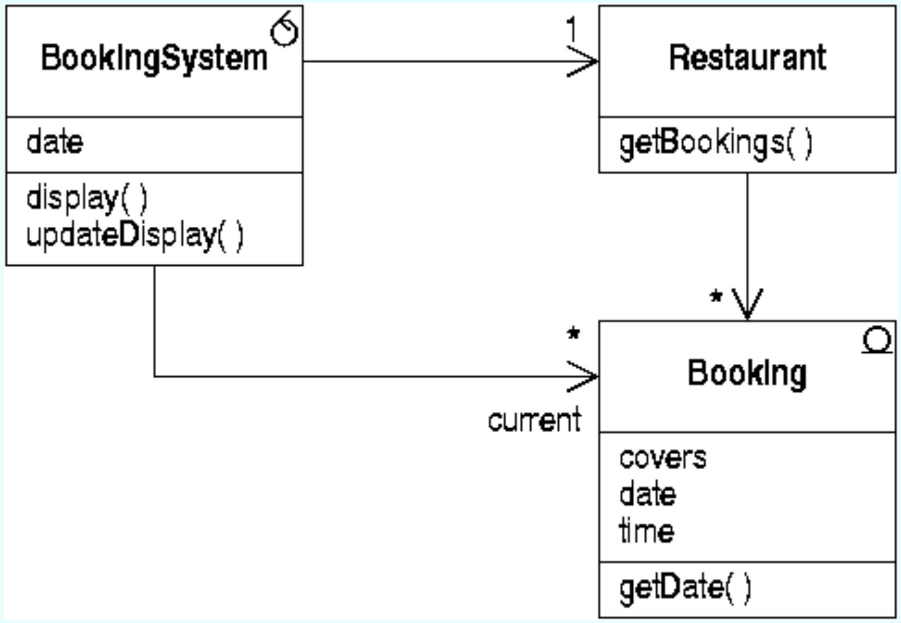
Refining the Domain Model

- This realization has involved:
  - new 'Restaurant' and 'BookingSystem' classes, with an association between them
  - an association from ‘Restaurant’ to ‘Booking’
    - ‘Restaurant’ maintains links to all bookings
    - messages sent from restaurant to bookings
  - an association from ‘BookingSystem’ to ‘Booking’
    - ‘BookingSystem’ maintains links to currently displayed bookings

49.55

Updated Class Diagram

- Operations are derived from messages sent *to* the instances of a class (fig. 5.6)

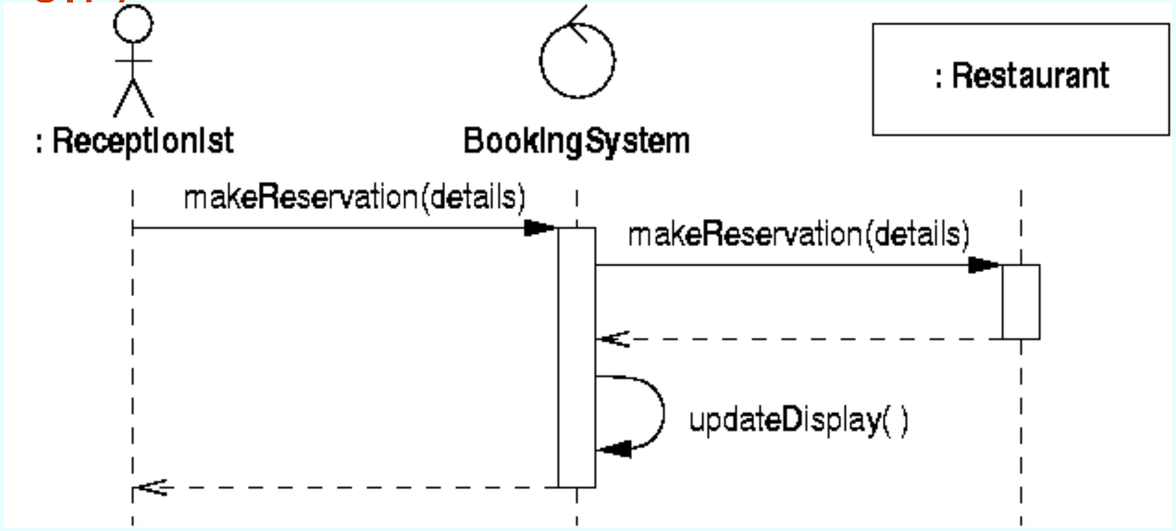


Mc Graw Hill Education

©The McGraw-Hill Companies, 2004

Recording New Bookings

- Give ‘Restaurant’ responsibility for creation
  - don’t model details of user input or data yet (fig. 5.7)

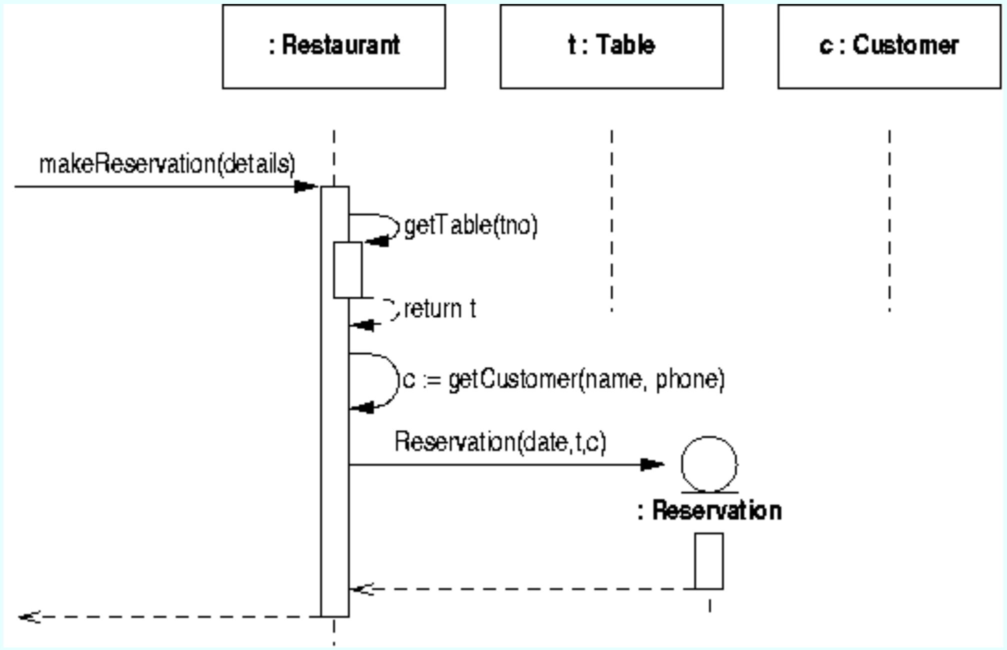


Creating a New Booking

- Bookings must be linked to table and customer objects
  - responsibility of ‘Restaurant’ to retrieve these, given identifying data in booking details
- New objects shown at point of creation
  - lifeline starts from that point
  - objects created by a message arriving at the instance (a *constructor*)

Creating a New Booking

- This completes the previous diagram (fig. 5.8)

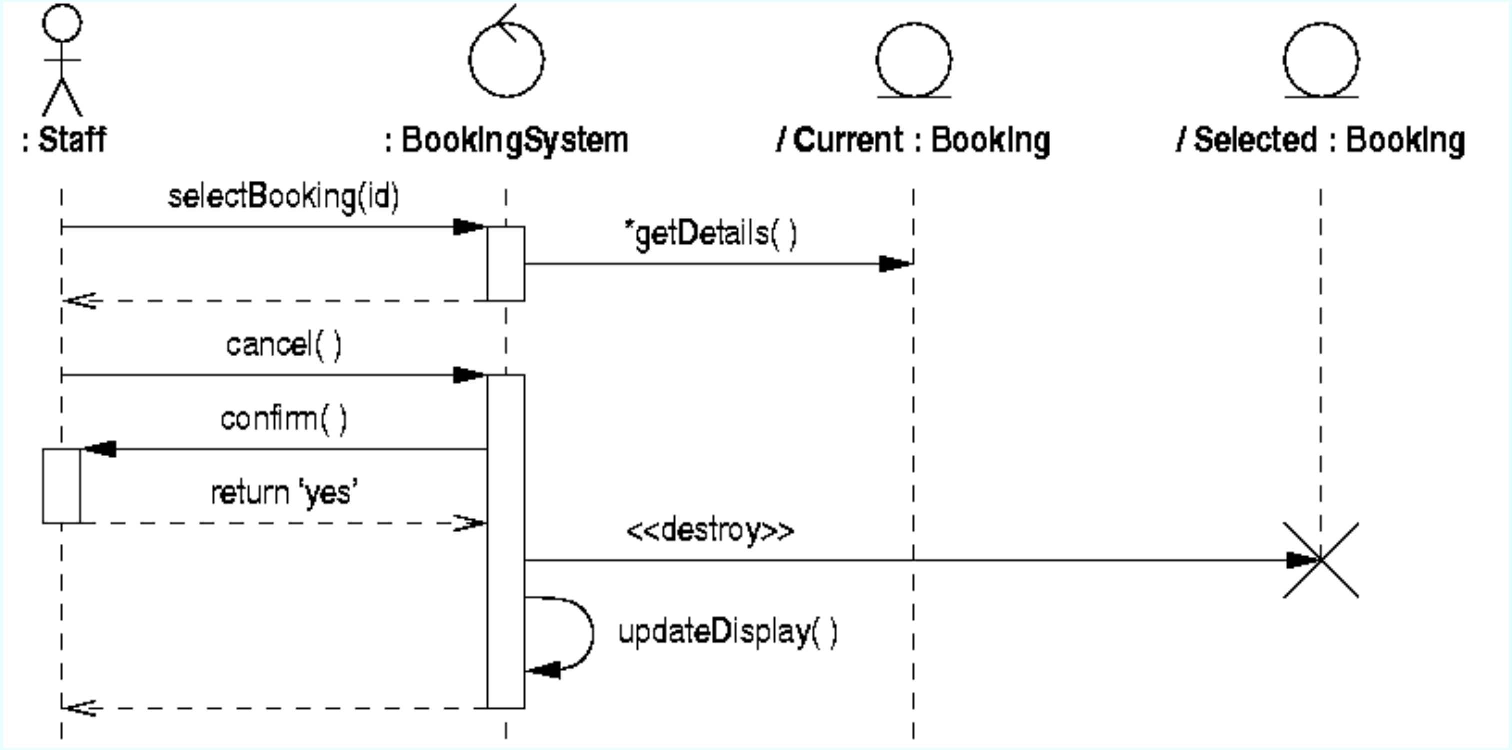


Cancelling a Booking

- A three-stage process:
  - select on screen the booking to be cancelled
  - confirm cancellation with user
  - delete the corresponding booking object
- Object deletion represented by a message with a ‘destroy’ stereotype
  - lifeline terminates with an ‘X’
- *Role names* used to distinguish selected object from others displayed

©The McGraw-Hill Companies, 2004

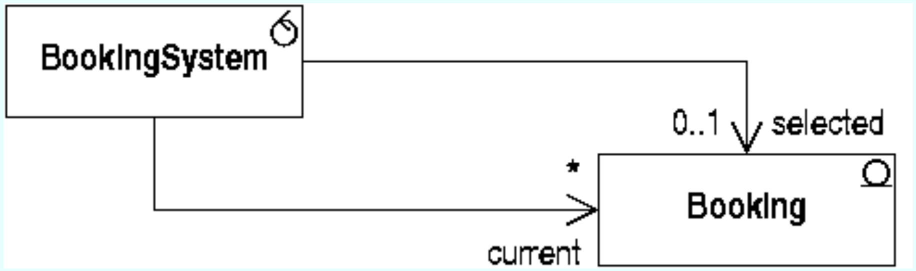
Cancelling a Booking (fig 5.9)



©The McGraw-Hill Companies, 2004

Refining the Domain Model (2)

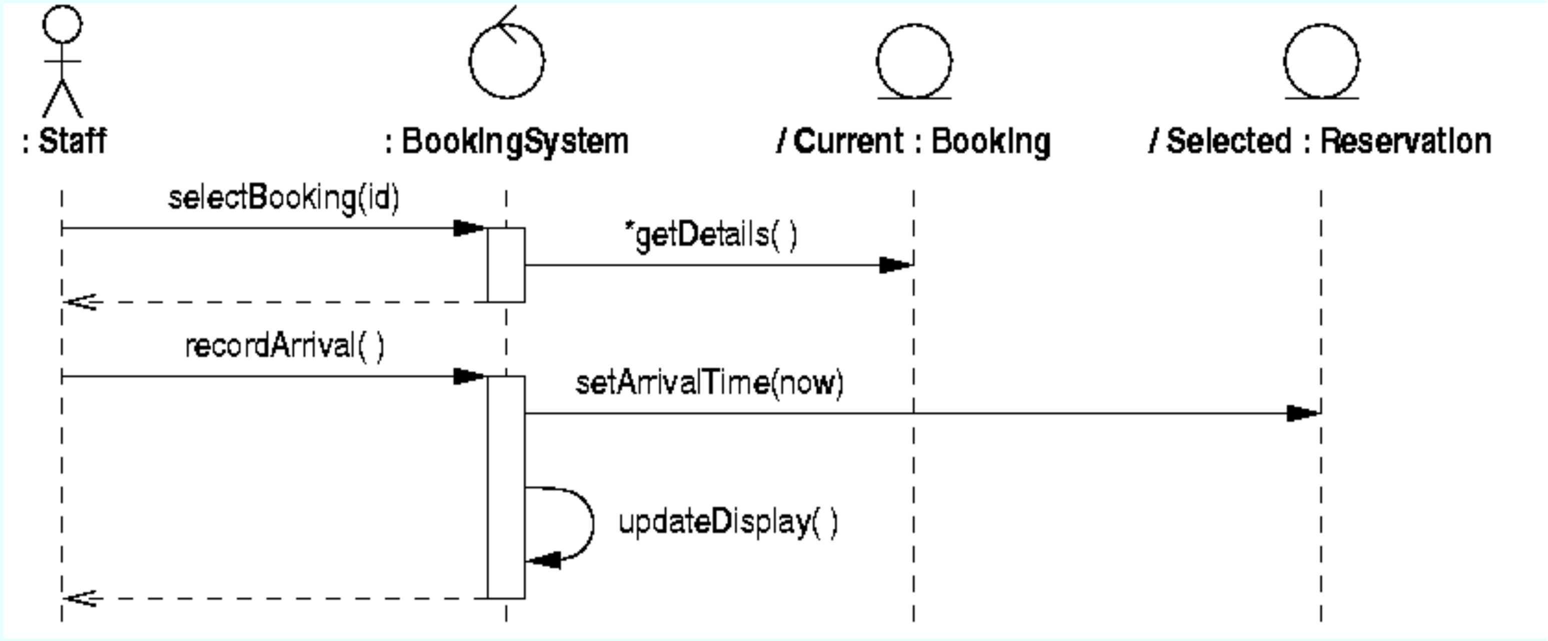
- ‘BookingSystem’ has the responsibility to remember which booking is selected
- Add an association to record this (Fig. 5.10)



©The McGraw-Hill Companies, 2004

Recording Arrival (5.11)

- Selected booking must be a reservation



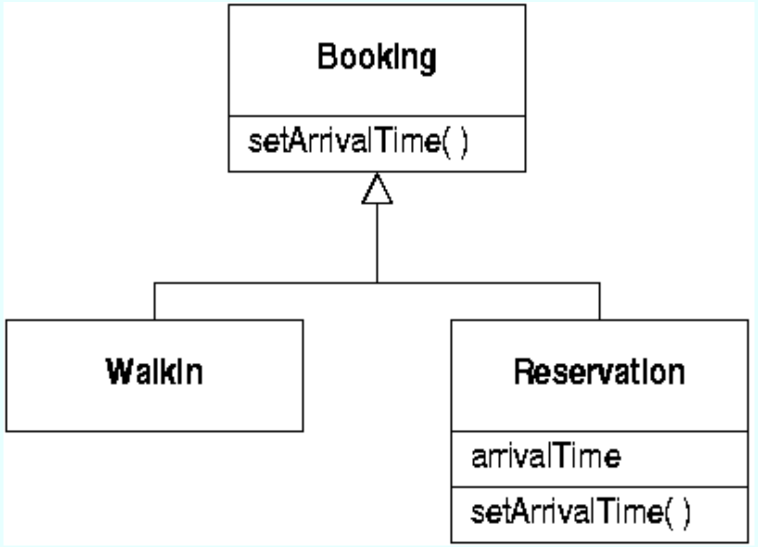
Mc Graw Hill Education

©The McGraw-Hill Companies, 2004

Class Interface Design

- Should ‘setArrivalTime’ be defined in Booking or Reservation class?
  - on the one hand, it doesn't apply to walk-ins
  - but we want to preserve a common interface to all bookings if possible
- Define operation in ‘Booking’ class
  - default implementation does nothing
  - override in ‘Reservation’ class

Refined Class Hierarchy (5.12)



Mc Graw Hill Education ©The McGraw-Hill Companies, 2004

Summary

- Analysis has led to:
  - a set of use case realizations
  - a refined class diagram
- We can see how the class design is going to support the functionality of the use cases
- This gives confidence that the overall design will work

Mc Graw Hill Education ©The McGraw-Hill Companies, 2004

Summary

- *Analysis* can be defined as the activity of representing the application domain and the system's requirement in terms of the object model
- The basic analysis technique is the production of *use case realizations*, which demonstrate how the functionality specified in a use case could be delivered by a set of interacting objects.

Summary

- *Realizations* can be documented using one of the forms of interaction diagram defined in UML i.e. collaboration or sequence diagrams.
- Producing use case realizations will suggest changes in the domain model, which will evolve into more detailed *analysis class model*.
- A central metaphor of object design is to make objects responsible for a subset of the data and operations in the system.

Summary

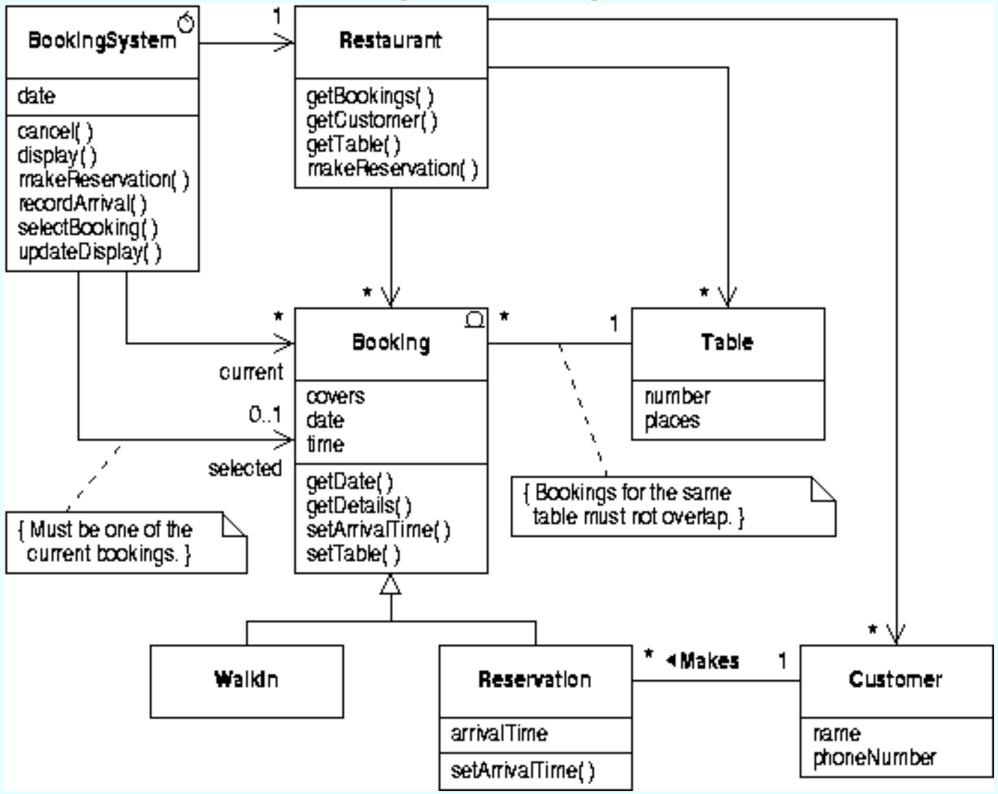
- The *Unified Process* includes an architectural description as one of the product analysis. A widely used architectural approach is to structure a system as a number of layers, for example presentation, application, and storage layer.
- The objects in a system can be assigned a number of roles, to clarify the organization of the system. UML defines class stereotypes *boundary*, *control* and *entity* objects

Summary

- User interaction can be shown on realizations by means of *system messages* received by control objects. There can be one control object per use case or one representing the system as a whole.

Mc Graw Hill Education ©The McGraw-Hill Companies, 2004

Complete Analysis Class Model (5.13)



©The McGraw-Hill Companies, 2004