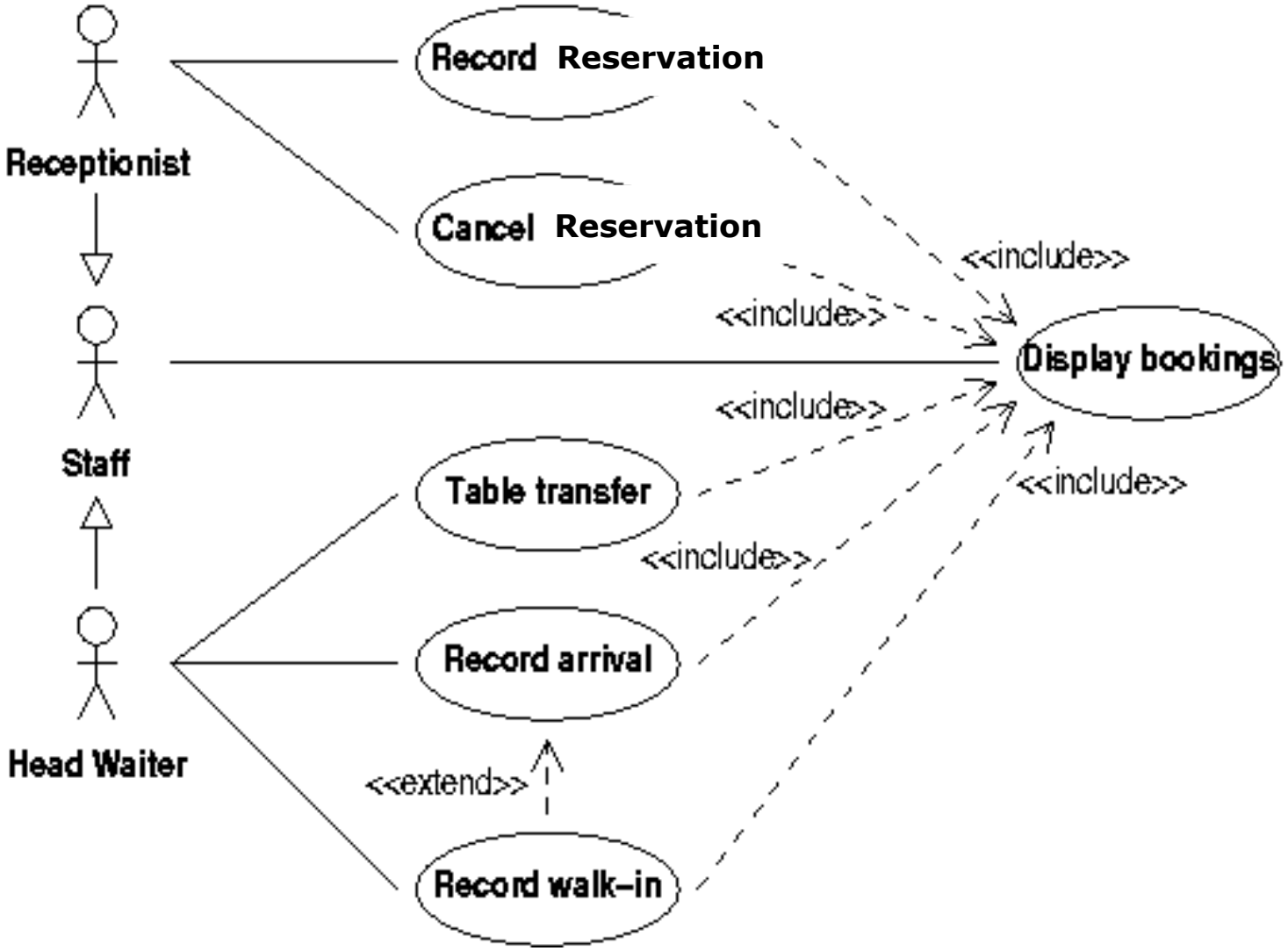
Table Booking System: Interaction Diagrams

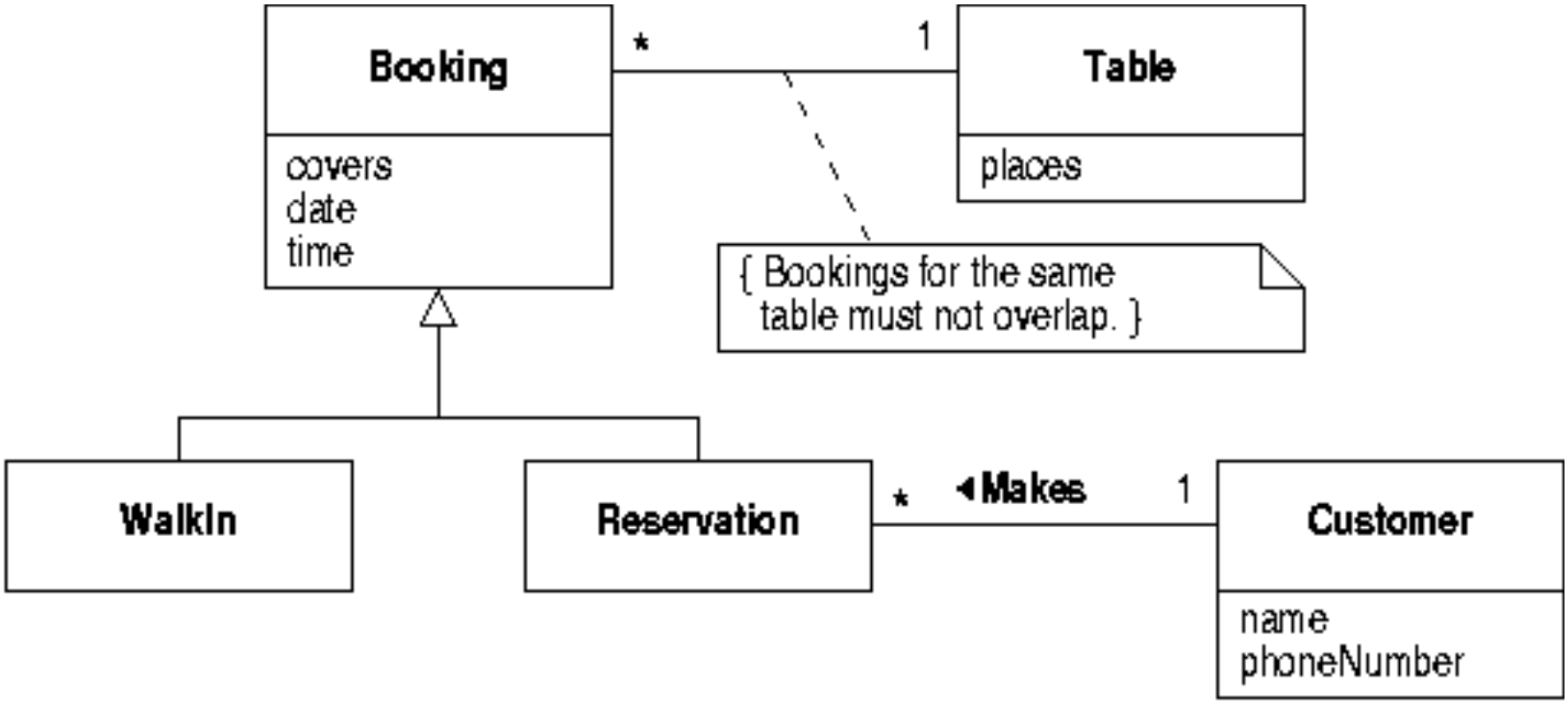
Comp 47480: Object Oriented Design

(Slides based on Chapter 5 of *Practical Object-Oriented Design with UML* by Mark Priestly)

Reminder: Table Booking Use Case Model



Reminder: Table Booking Domain Model



Interaction Diagrams

What is to be analysed?

the Use Cases

There are several types of Interaction Diagram; we look only at the **Sequence Diagram**.

Why?

to demonstrate that they can be implemented using the classes identified in the Domain Model (plus probably a few more)

How?

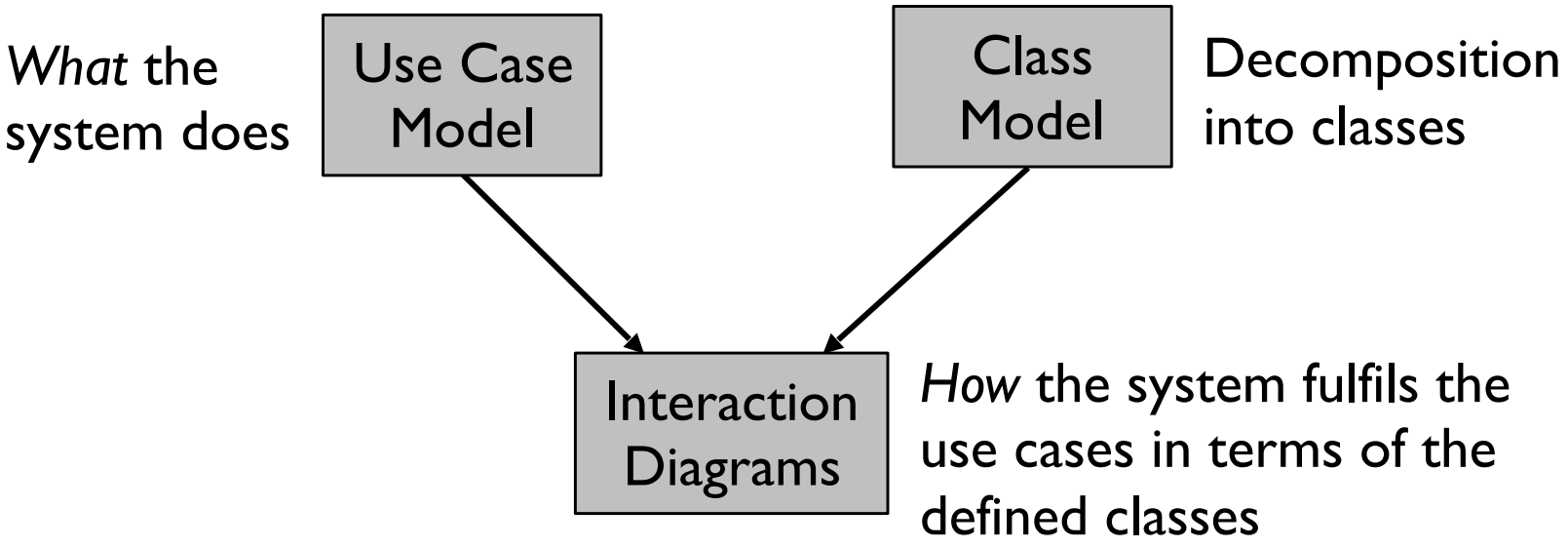
by creating Interaction Diagrams that *realise* the Use Cases

We need to show the relationship between the Use Case model and the Class model:

How can objects of the identified classes collaborate in order to provide the functionality described by the use cases?

Relationship between the Models

The relationship between the models we have seen can be visualised as follows:



Remember that the development process is iterative. Each model can bring an understanding that causes us to revisit another model.

Is it Analysis or Design?

- Difficult to draw a clear boundary. Traditional informal distinction:
  - analysis models the real-world system (the 'what')
  - design models the software (the 'how')

- Object-oriented methods use the same notation for both activities
  - encourages 'seamless development' and iteration
  - analysis/design gap not so clear

Designing our Classes

- We ultimately want to define attributes and methods for each class in the model.

- The domain model alone is insufficient because
  1. structure of a real-world application is probably not the optimal structure for a software system
  2. domain model does not show methods

- *Realisation* of the use cases as Interaction Diagrams identifies methods and confirms that the design supports the required functionality

Warning!

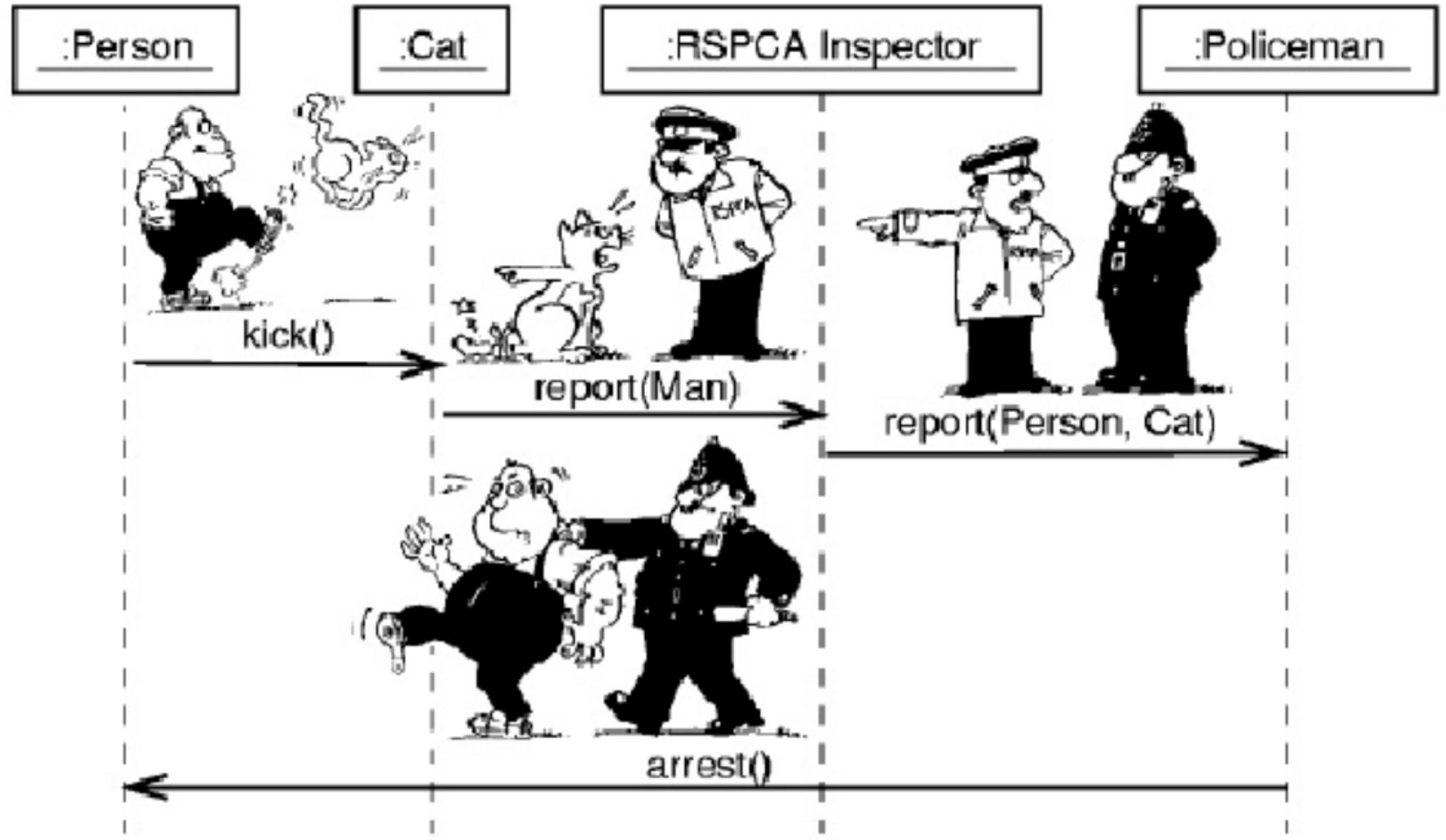
A UML purist might say that each use case should have a set of interaction diagrams to capture its normal, alternate and exceptional scenarios.

This is probably a waste of time. Unless creating the interaction diagram feels useful, increases your understanding and helps resolve some design issues, don't do it!

If the interaction seems obvious and you're confident you could just code it up, do just that.

Especially don't model UI elements and interactions that will simply be provided by some implementation framework.

Sample Interaction



Realising the 'Display Bookings' Use Case

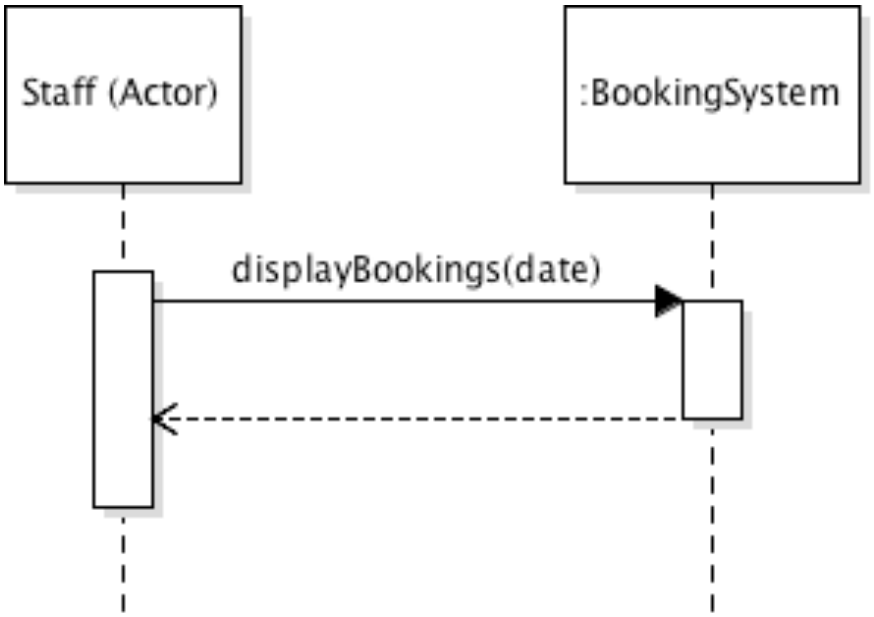
- The 'Display Bookings' use case has a simple dialogue
  - the user provides the required date
  - the system response is to update the display

- Our initial realisation consists of
  - instance of the 'Staff' actor
  - an object representing the system (BookingSystem)
    - matter of taste, not necessary
  - message(s) passed between them

- Beware of the BookingSystem object becoming a God class! If it does, split it.

Actor interaction with 'Display Bookings'

- System messages are sent by an external actor
- Note that we're analysing **use case behaviour**, not UI design



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

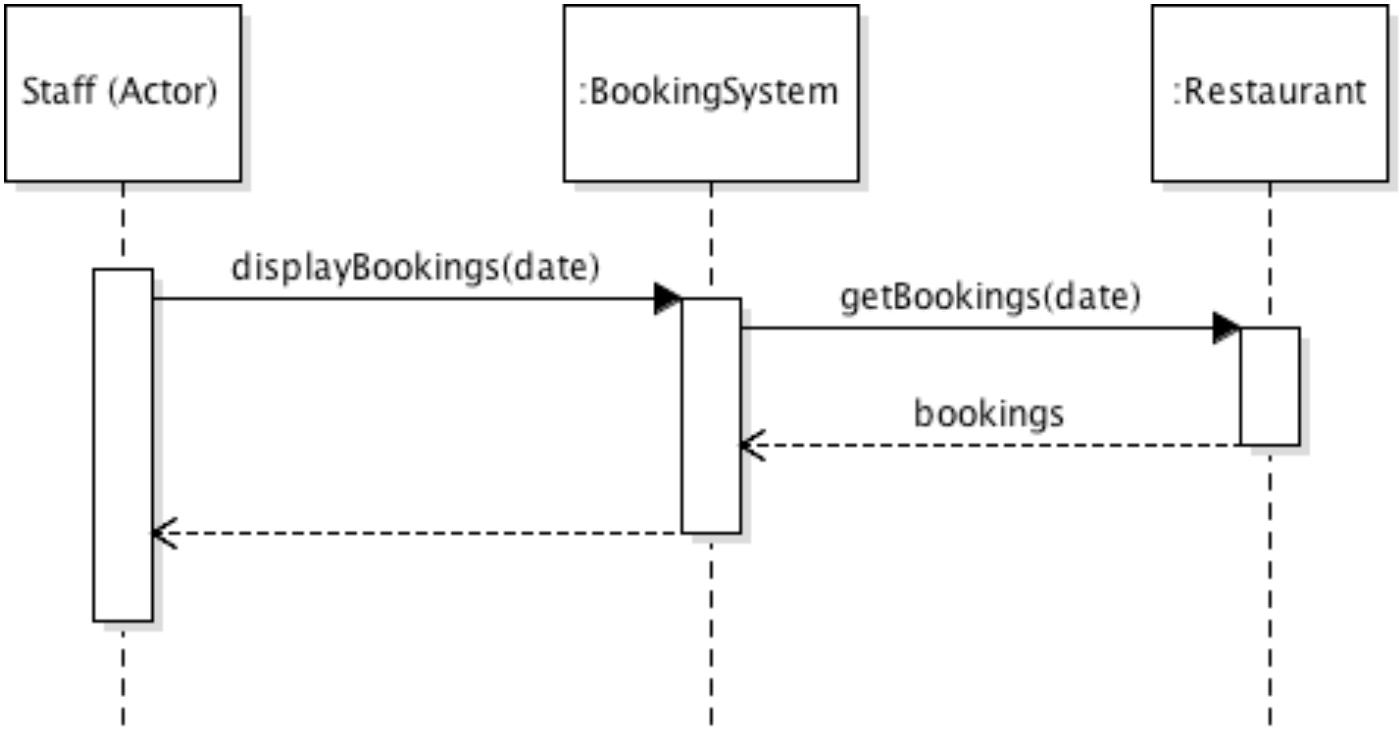
Accessing Bookings

- 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
  - it makes no sense in the 'Booking' class either
  - so we define a new 'Restaurant' object with the responsibility to manage booking data

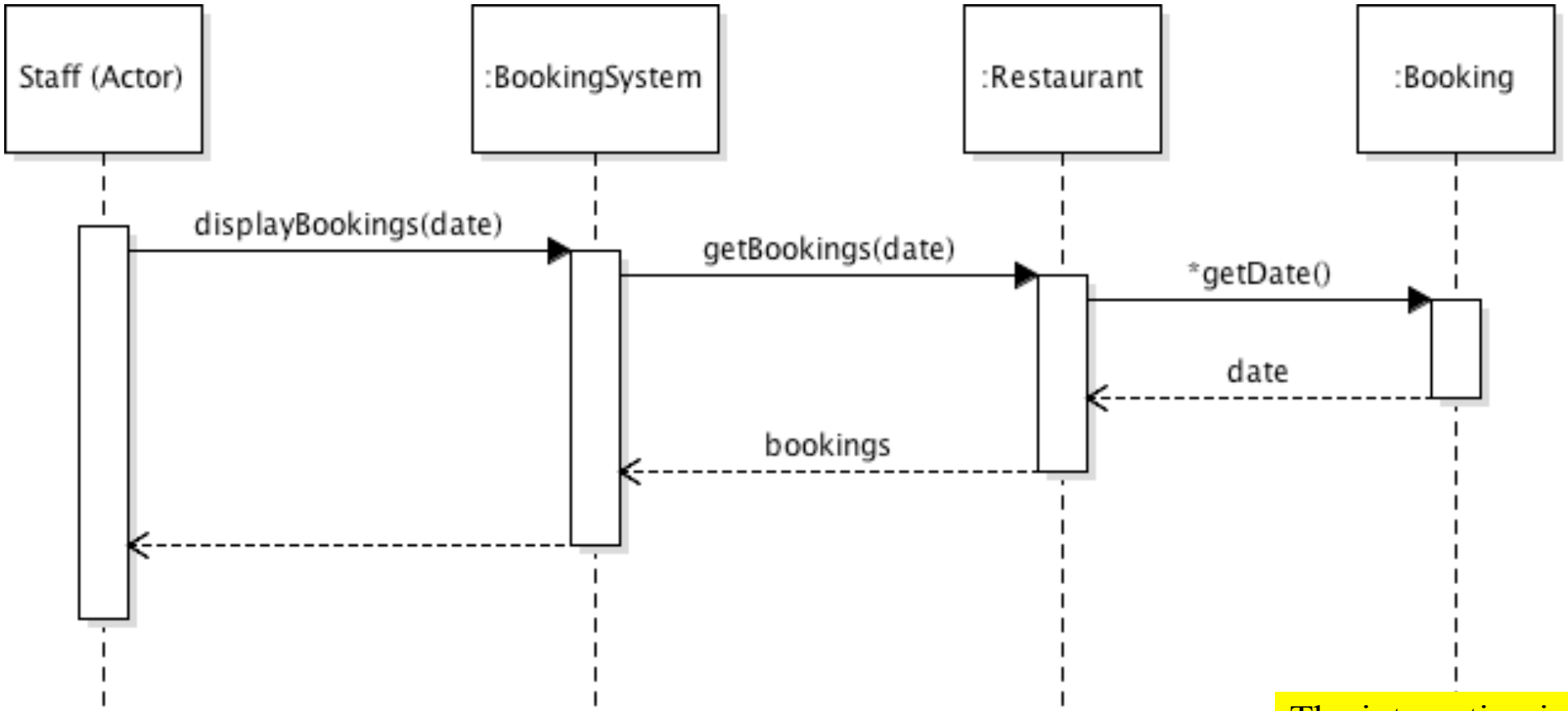
Retrieving Bookings

- Add a message to get relevant bookings



Retrieving Booking Details

- Dates of individual bookings will need to be checked by the 'Restaurant' object



The interaction is getting very detailed here: I'd leave this to be solved in code.

Refining the Domain Model

- This realisation causes us to revise the class model in a number of ways.

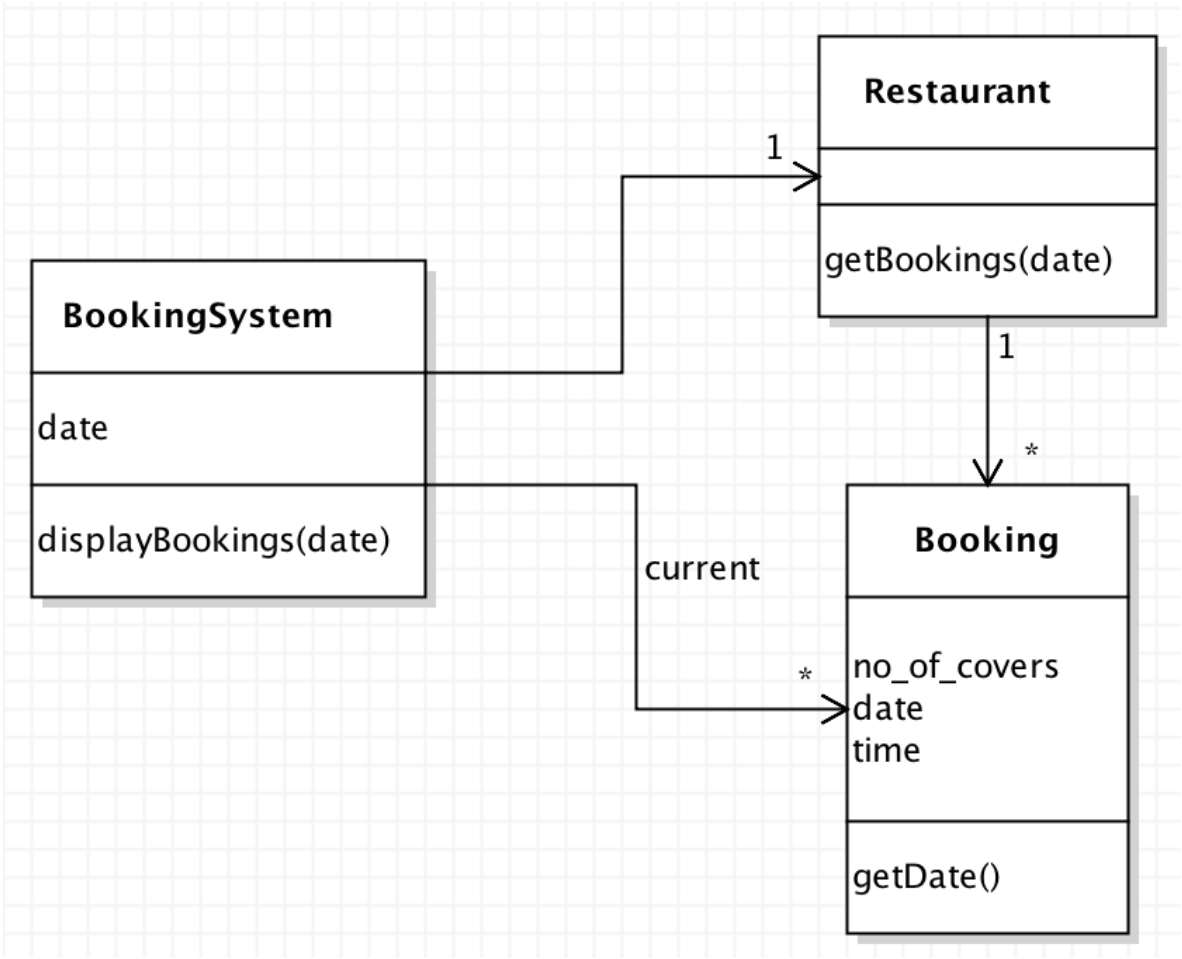
- new 'Restaurant' and 'BookingSystem' classes, with an association between them

- 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

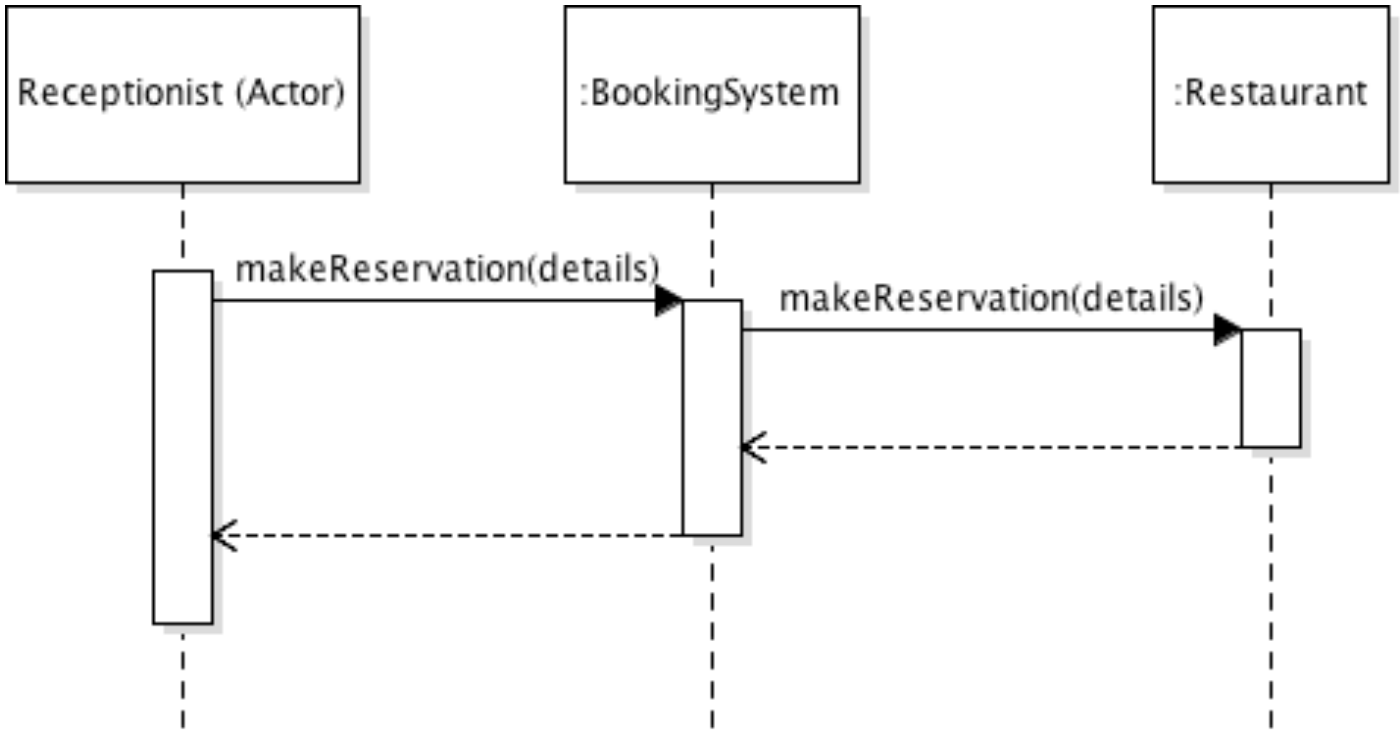
Updated Class Diagram

- Operations (methods) are derived from messages sent to the instances of a class.
- Note Navigation arrows. Messages can only be sent in the direction of the arrow.



Another Use Case: Recording New Reservation

- Give 'Restaurant' responsibility for creation
  - don't model details of user input or data yet



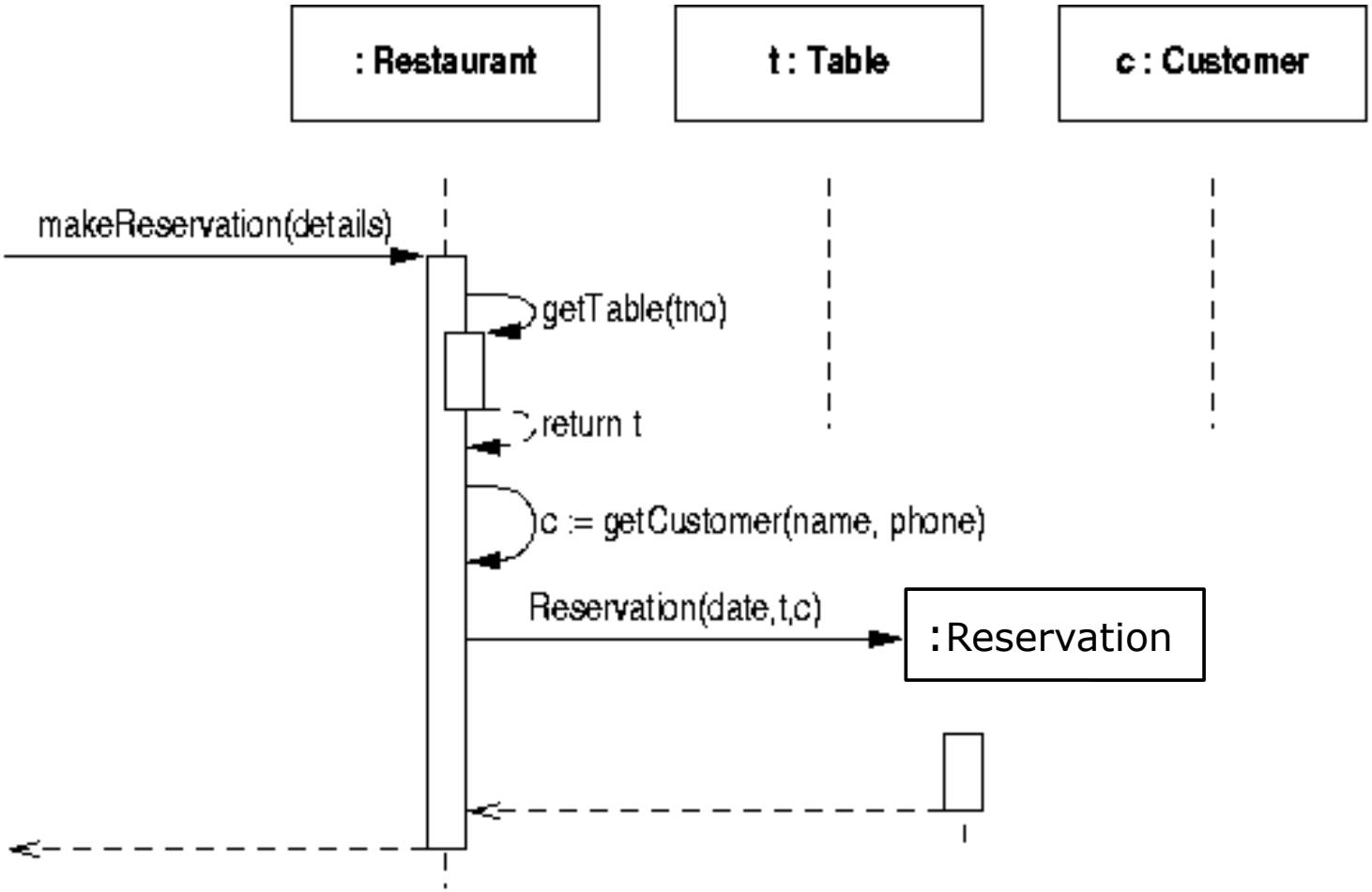
Creating a New Reservation

- Reservations must be linked to table and customer objects
  - Actor provides table and customer numbers
    - Not ideal! The system should really propose a table.
  - responsibility of the Restaurant object to retrieve these, given identifying data in reservation 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 Reservation

Note Object Creation

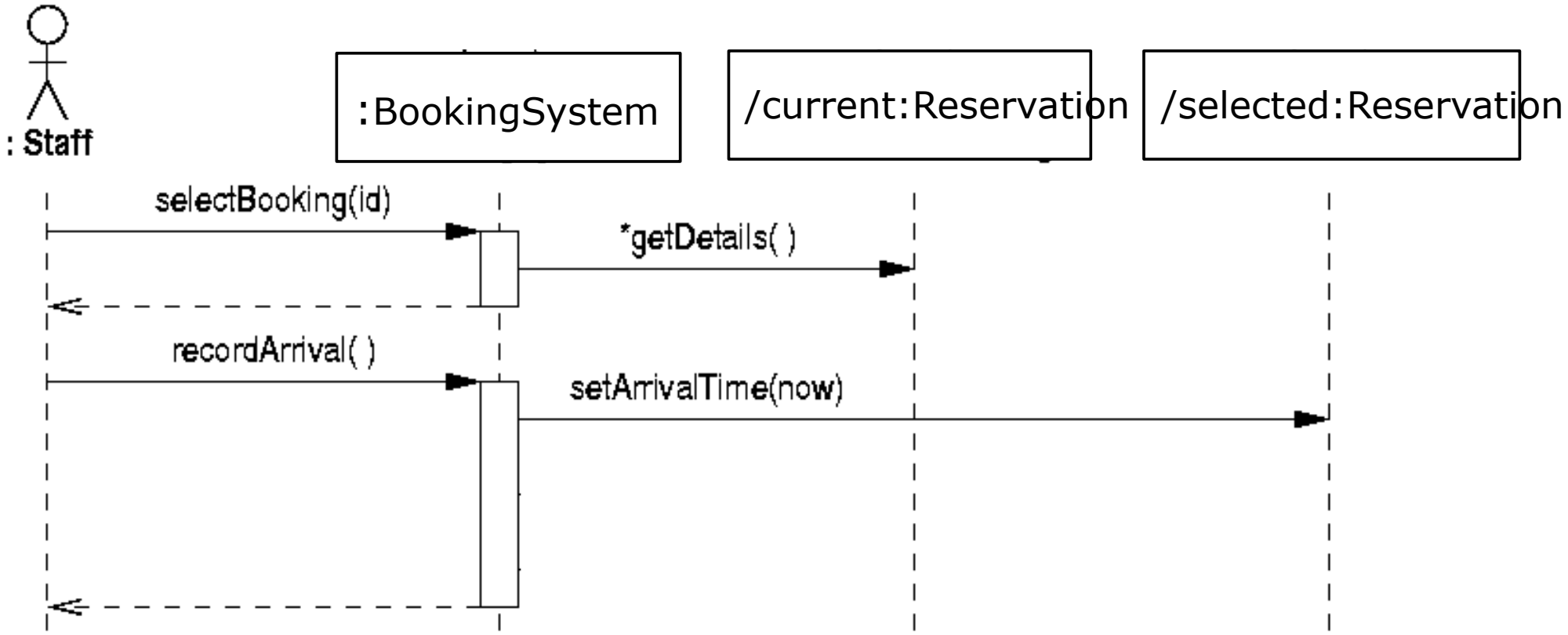
- This completes the previous diagram
- Note two ways of showing 'message send to self'



Use Case: Recording Arrival

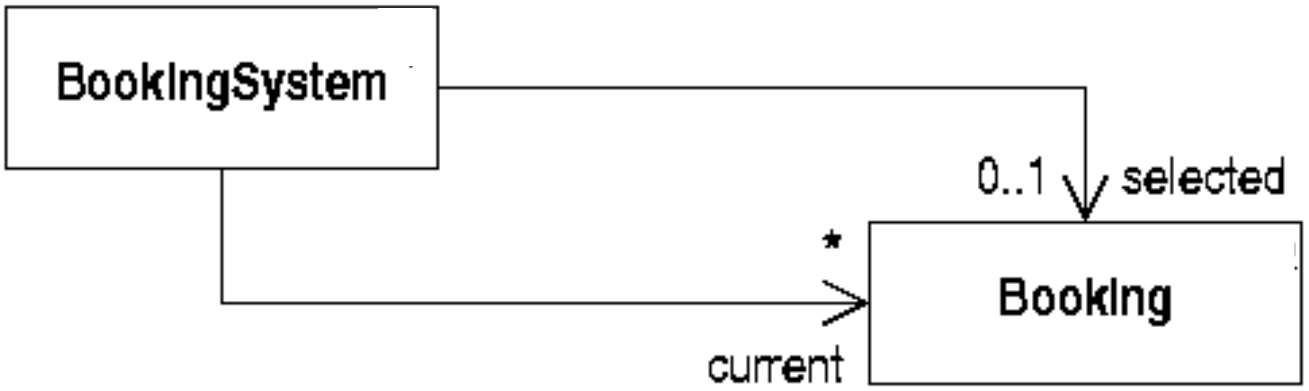
Note role names

- Selected Booking must be a Reservation (not a Walk-in)



Refining the Domain Model Again

- 'BookingSystem' maintains a list of current bookings that are displayed to the user, one of which booking is the selected booking.
- We add an association to record this



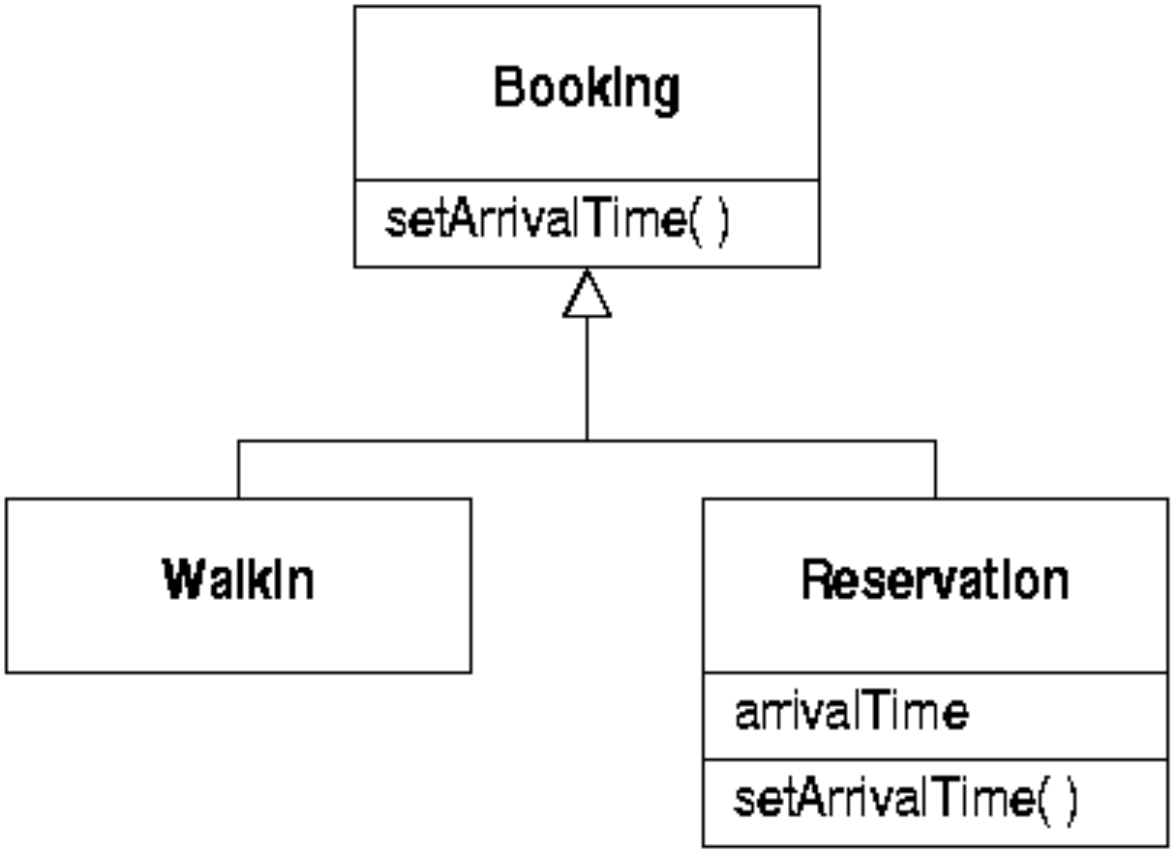
Class Interface Design

Common type of design decision

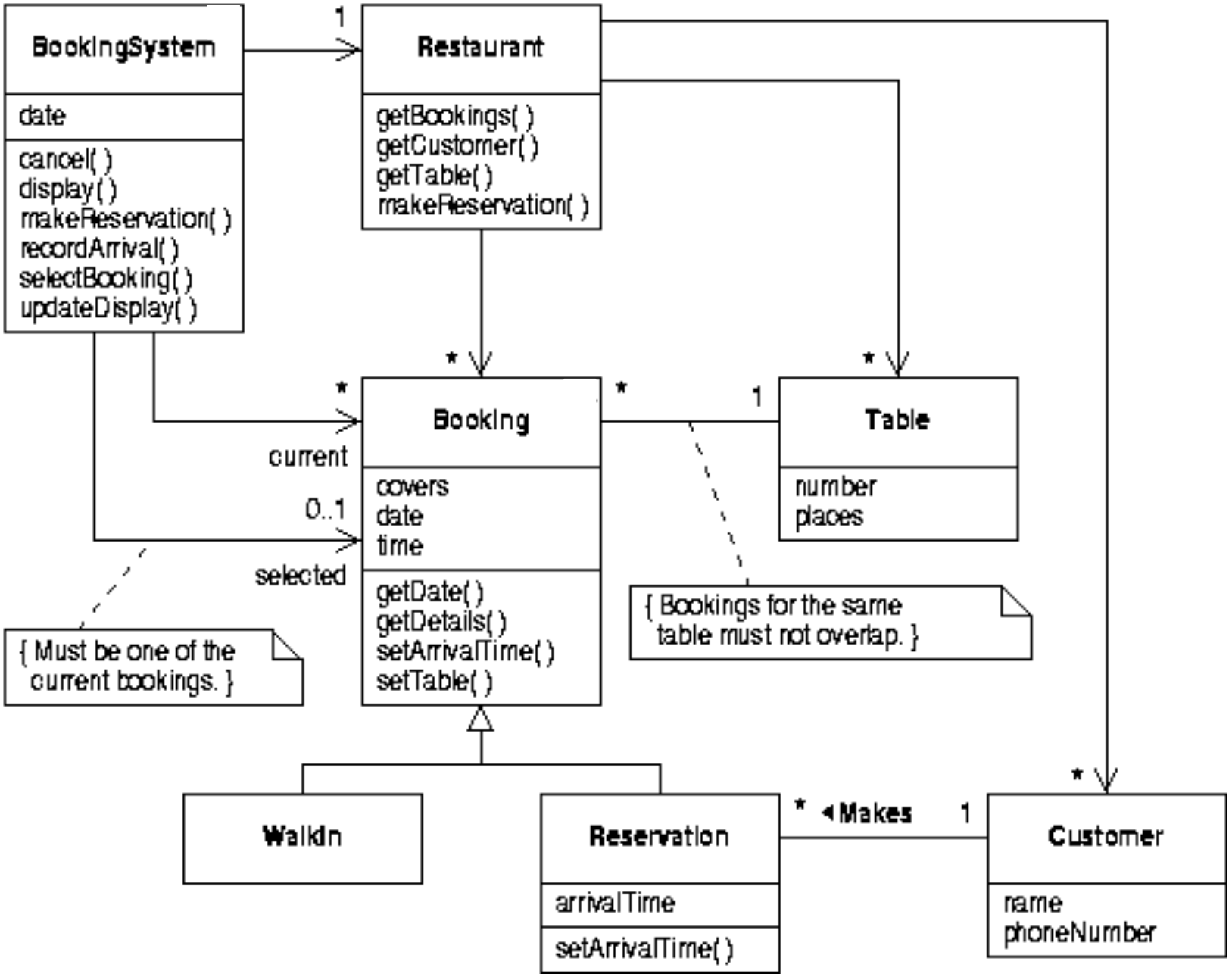
- Should setArrivalTime be defined in Booking or Reservation class?
  - on the one hand, it doesn't apply to Walk-ins
  - but we want to preserve a common interface to all bookings if possible

- Probably best to define setArrivalTime in Booking class
  - default implementation does nothing
  - override in Reservation class

Refined Class Hierarchy



Updated Class Model



Summary

- Our analysis has led to:
  - a set of use case realisations
  - a refined class diagram
- We can see better how the class design is going to support the functionality of the use cases
- This gives confidence that the overall design will work
- In general, whenever objects have an interesting interaction, it may be useful to use an Interaction Diagram to capture it.