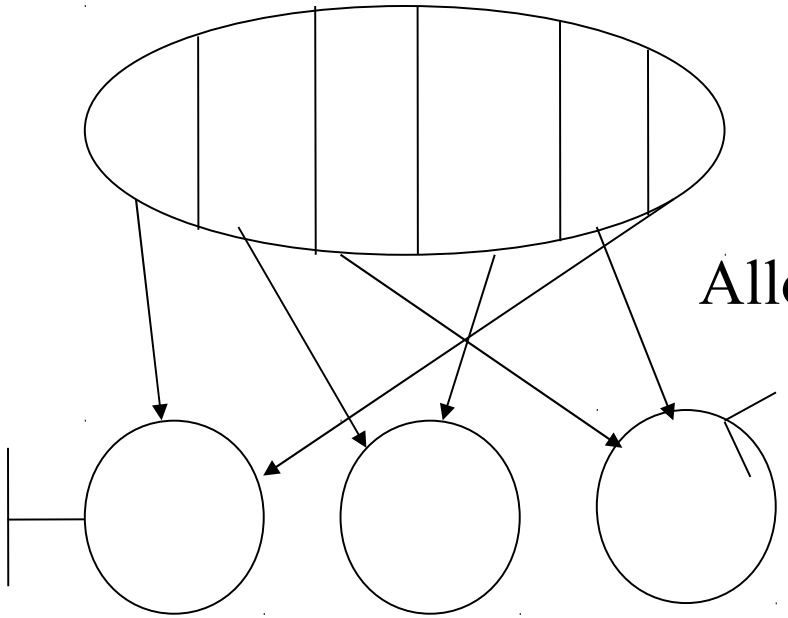
Finding Classes

- Aim: Elicit the classes for our class diagram from the use-case descriptions.
- The ‘expert’ views
  - Eriksson and Penker
  - Pooley and Stevens
  - Booch, Jacobson and Rumbaugh (the three amigos) say…
    - ‘Use cases ‘reflect rather than specify the implementation of a system, subsystem or class.’

Classes from Use Cases: OOSE



Interface objects

Entity objects

Control objects

Functionality of a use case

Allocation to object responsibilities...

...but doesn't say how.

Jacobson's OOSE Approach

A Method?

- Perceived wisdom in eliciting classes is..
  - Think like an object.
  - Search for nouns (classes), verbs, etc..
  - CRC cards,
  - Business analysis / process model.
  - Get the user to tell you (if they know).
- However, use cases are supposed to aid comprehension.
  - and comprehension suggests...

Questions

- What Questions should we ask of a description.
  - Ones that allow us to specify / design?
  - Some to identify the objects, attributes, services we need. (More intuitive - next).
  - Some to test the behaviour / dependencies of actions. (Which is usually implicit).
    - May use other models to aid this process.
  - Some to check the communicability.
  - Some to assess the 4Cs.

What do we need to find?
where they go comes later

- The roles or actors
- The classes or objects stated.
  - Their attributes and services.
  - The objects, services & attributes not stated (but implicit) or suggested.
  - The inter-dependencies among roles (actors) and objects.
  - The correct sequence of actions (events)...
    - and their pre / post conditions.
- Any problems inconsistencies or assumptions.

Four C's of Communicability Consider (don't ask specifically)

- Coverage
  - Complete, Rational, Span, Scope
- Coherence
  - Logical order, Logical coherence, Consistent Abstraction
- Consistent Structure
  - Variations,Grammar, Sequence
- Consideration of Alternatives
  - Separation, Viability, Numbering

An Example

- Work through the car-park example
  - Well Use Case One.
- Ask ‘standard’ or generic questions.
  - What we need to know.
  - Comprehension type questions.
- Ask questions specific to the description.
  - Dependency and association type questions.

Will ask questions about..

- Dependencies
- Interface
- Actors
- System
- Classes & Objects

- First the Use cases...

Use Case 1: Enter Car Park

- **Actors:** Driver
- **Context:** The Driver wants to park in the local "Regional Car Park" so the Driver can go shopping.
- **Pre-condition:** There are parking spaces available inside the car park.
  - *(How do we know?)*
- **Post condition:** There is one less space available inside the car park.

Main flow of events:

1. The *Driver* drives to the ticket machine.
2. The *Driver* presses the ticket button.
3. The *ticket machine* dispenses a ticket.
4. The *Driver* takes the ticket.
5. The *entry barrier* raises.
6. The *Driver* drives into the car park.
7. The *entry barrier* lowers.
8. The *Driver* parks the car.

Exceptional flow of events:

3. The ticket machine fails to dispense a ticket. The Driver calls for assistance.

Questions: Dependencies

- Dependencies (pre- and post-conditions)
  - Each event and resulting action in the use case is dependent upon what?
  - What dependencies do you have to assume, and where?
  - Do you have to assume there are other actors involved in the system?
    - that are necessary to assure dependencies,
    - but not stated in the use case.

Behaviour Considered

- Consider / walk through the process or scenario.
  - Use Case 1 has explicit (active) objects / actions… (with some inter-dependencies).
    - Driver: drives to.., presses.., takes.., drives.., parks.
    - Ticket machine: dispenses..,
    - Barrier: raises.., lowers..
  - and some implicit (passive) objects ..
    - Ticket. (Sensors?) (Sign?) (Car park or car count)
  - Consider inter-dependencies in more detail.

Dependencies: Informal

- *The driver approaching sees sign "Spaces Inside".*
  - *This event is dependent on there being spaces available.*
  - *That is, the sign has not been informed that the car park is full.*
  - *Who informs the car park?*
  - *There must be an object that knows about the number of cars in the car park.*
- *Issuing of a ticket by the ticket machine is dependent upon:*
  - *space available, a ticket dispenser receiving a signal to print, the ticket dispenser having an internal clock that prints time and date on the ticket.*
  - *Post-condition is that the ticket machine presents the driver with a valid ticket to take.*

Dependencies: Barriers

- *The barrier raising is dependent upon the barrier knowing when the car is allowed to approach.*
  - *Assume that once the driver has pulled the ticket from the machine, the barrier is told that it should raise.*
- *The barrier lowers only when the car has driven under it, we assume.*
  - *Might be a timing mechanism that keeps the barrier up for 30 seconds, for instance.*
  - *Likely there is a sensor or pressure pad that senses when the car has safely come past.*
  - *This sensor (or other object) then signals the entry barrier to lower.*

Formal Models of Behaviour and Dependencies

- Consider all roles / objects acting in parallel.
  - For each event of a role / actor / object consider its state before and after the event.
  - Consider what other objects are involved in the event (interaction) and their pre / post conditions.
  - The pre / post conditions will then control the synchronisation of all of the roles / objects / actors in the process.

Consider an event or action

***Pre**: Driver not at machine (initial).*

Both machine and driver in this (initial) state.

How did driver know there were spaces? (Sign + Object that knows count)

- The *Driver* drives to the ticket machine.

***Post**: Driver (at Ticket Machine).*

Both machine and driver in this new (at machine) state.

Ticket machine notified of this by what object? (Sensors)

Hidden action(s)

1.1 The *Driver* drives over the entry pad (sensor).

Pre and post as for ticket machine

Me (initial -> overPad)

EntryPad(initial -> overPad)

1.2 The *Sensor* notifies the ticket machine [of what…?]

```
Selection Driver.driveOverPad
   Me( initial -> DriverAtMachine )
   EntryPad( initial -> overPad )
End

Selection EntryPad.PadNotify
   Me( overPad -> initial )
   TicketMachine( initial -> CarAtMachine)
End

Selection Driver.PressForTicket
   Me( DriverAtMachine  -> ticketRequested )
   TicketMachine( CarAtMachine -> ticketRequested )
End

Selection TicketMachine.Dispense
   Me( ticketRequest -> ticketDispensed )
   Ticket ( initial -> date_stamped )
End

Interaction Driver.TakeTicket
   Me( ticketRequested -> ticketTaken)
   TicketMachine( ticketDispensed -> ticketTaken )
```

Object States: Formal

- Dependencies for 1 to 4.
- States act as pre / post conditions.
- E.,g., for driver to take ***ticket*** it must have been dispensed.
- Ticket not from behaviour, but a data object.
  - and so on...

```
Selection TicketMachine.NotifyBarrier
   Me( ticketTaken -> raisedBarrier )
   EntryBarrier( initial -> raised )
End

Selection Driver.driveOverPad2
   Me( ticketTaken -> DriverInCPark )
   EntryPad2( initial -> overPad )
End

Selection EntryPad2.PadNotify
   Me( overPad -> initial )
   TicketMachine( raisedBarrier -> CarInCPark)
End

Interation TicketMachine.NotifyBarrier
   Me( CarInCPark -> initial )
   EntryBarrier( raised -> initial )
End

Action Driver.Park
   Me ( DriverInCPark -> Parked)
End
```

Further Events

- Forces questions.
  - After barrier up (5), how do we know a car has gone through (6), in order to initiate (7) lower barrier.
    - Another pad (after the barrier) or a timer?

Questions: Interface

- Is there any direct interaction between actor and system? Where does this occur?
  - What kind of interaction is this? E.g. completing a field at an interface, pressing a button on a machine, speaking to a system and getting a verbal response or seeing something at a visual level of communication?
- Does the use case impose any style of interface design?
- Does the use case suppose any style of interface design?

Interface: Assumptions

- *The actor (driver) hits the button on the ticket machine to dispense a ticket.*
  - *a physical contact between the driver and system.*
  - *constraining design here*
  - *a ticket could be automatically dispensed*
- *The system dispenses a ticket to the driver.*
  - *The driver takes this from the system.*
  - *We are not informed by what means the system dispenses the ticket (a little slot or a bucket).*
  - *Might be assuming a specific design because the system has to indicate to the barrier when to raise.*

Questions: Actors

- Are there actors missing from the use case?
  - how and where do they link to the system?
- What assumptions are made about the actors at the interface?
- Is the actor acting as a "go-between" between the system and someone else?
  - If so, what do we know about the customer who links with the actor at the interface?
- Are any of the actors other systems?
  - Are they passive or active?

Car Park Actors

- *No other actors are required for this use case.*
  - *(If there were exceptions we had to deal with then maybe we'd require other actors, e.g. if the ticket machine ran out of tickets perhaps there is a maintenance man to fix this problem.)*
  - *Attendants tend to act as checking mechanisms.*
- *Assume that the driver can reach the ticket machine easily.*
- *The driver does not act as a go between.*
  - *We don't need to know anything particular about the driver here.*
  - *HINT: No data object for driver.*
- *There are no other systems that interact with the car park.*
  - *At least not yet (or that we know of).*

Questions: System

- What levels of abstraction are shown in the use case for the system?
  - Does the use case describe
    - responses to actor actions that are visible to the actor (that is, interface responses)?
    - actions that are internal to the system (not visible to the actors)?
- What assumptions are made about system actions?
  - What do we assume the system has to do, so that the action taken by the system at the interface is valid?
  - Where in do we have to make those assumptions?

Car Park System

- Two levels of abstraction shown
  - what the driver does
  - and the interface interactions between driver and system.
- There is no system internal description.
  - Use cases don't typically do this.
  - Hence, we have to make assumptions.

System Assumptions

- *Sign knows when / what to illuminate.*
  - *Informant must be an object that knows about the number of cars in the car park.*
- *Ticket machine working & must have an internal clock.*
  - *Assume that the ticket machine sends a message to the entry barrier to raise (on ticket taken).*
- *Barrier lowers when pad informs (which object?)that car has safely come past.*
  - *Another object (the one informed) controls the lowering of the barrier.*

Classes (and Objects)

- What are the classes in the use case? Look for nouns/names in the use case.
  - Are there classes you have to “invent”?
  - That is, they are implied but not explicitly stated.
- What operations associate classes with others?
  - Often identified as verbs linking nouns (nouns already identified as classes).
- What attributes are identified for each class?
  - What operations do the classes perform on themselves or for other classes?

- *Class: Driver.*
  - *Operations: press Ticket Machine button, take Ticket, drive car.*
  - *Attributes:*
- *Class: Ticket Machine.*
  - *Operations: dispense Ticket, signal Entry Barrier to raise.*
  - *Attributes:*
- *Class: Ticket.*
  - *Operations:*
  - *Attributes: time, date.*
- *Class: Entry Barrier.*
  - *Operations: raise, lower.*
  - *Attributes:*

Classes

- *Class: Car Park Counter (implied)*
  - *Operations: decrement spaces count, signal Sign*
  - *Attributes: spaces count, max spaces*
- *Class: Sign (implied)*
  - *Operations: switch on*
  - *Attributes: on/off*
- *Class: Sensor (implied)*
  - *Operations: sense, signal Entry Barrier to lower, signal Car Park Counter to decrement*

Design

- Of course all this elicitation does not tell us where to put all services and attributes (which objects). But it does help.
- Similarly some of the dependencies imply associations.
- However, we still need a good deal of creativity to produce an OOA or OOD.
- Need to consider the other use cases.
  - Typically many and a Use Case Diagram.

Use Case 2: Exit Car Park

Main flow of events:

1. The Driver drives to the exit barrier.
2. The Driver hands the car park ticket to the Car Park Attendant.
3. The Car Park Attendant checks the ticket against a tariff list.
4. The Car Park Attendant informs the Driver of how much to pay.
5. The Driver pays the Car Park Attendant.
6. The Car Park Attendant raises the exit barrier.
7. The Driver drives out of the car park.
8. The exit barrier lowers.

Exceptional flow of events:

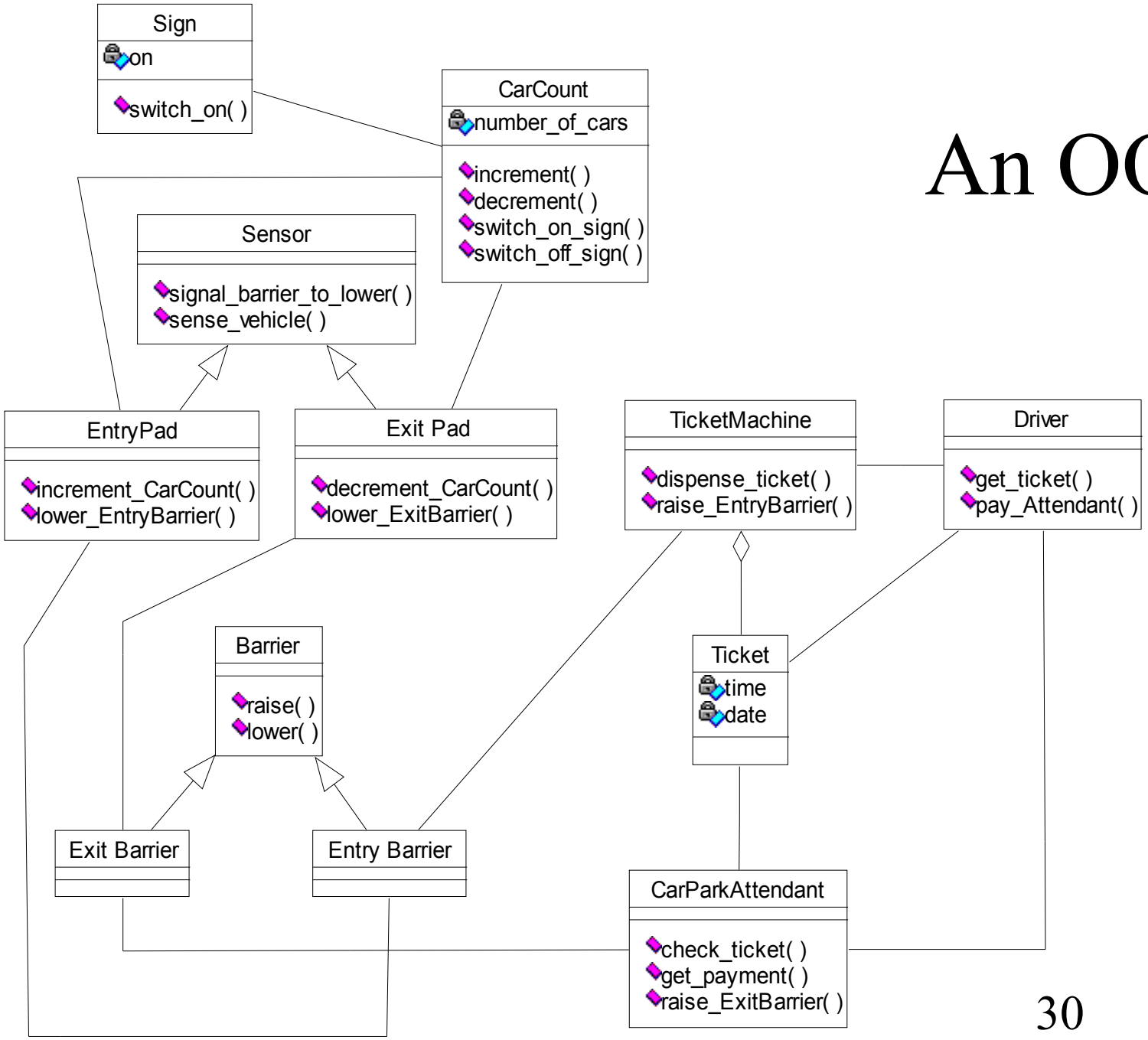
2. The Driver has no car park ticket. The Car Park Attendant charges a standard fee.
5. The Driver has no money. The Car Park Attendant impounds vehicle.

- Actors: Driver, Car Park Attendant.
- Context: The Driver wants to leave the car park.
- Pre-condition:
- Post-condition: There is one more space available in the car park.

Further Design

- Still need to apply our object oriented principles and to look for:
  - Hierarchy - classes and subclass (abstract classes - sensors and barriers.
  - Part of relationships (or something close).
  - Associations.
- Hence, in summary.
  - Questions / method may help to tease out information, but creativity still required.

An OOA


Sign
on
switch_on( )
CarCount
number_of_cars
increment( )
decrement( )
switch_on_sign( )
switch_off_sign( )
Sensor
signal_barrier_to_lower( )
sense_vehicle( )
EntryPad
increment_CarCount( )
lower_EntryBarrier( )
Exit Pad
decrement_CarCount( )
lower_ExitBarrier( )
TicketMachine
dispense_ticket( )
raise_EntryBarrier( )
Driver
get_ticket( )
pay_Attendant( )
Barrier
raise( )
lower( )
Ticket
time
date
Exit Barrier
Entry Barrier
CarParkAttendant
check_ticket( )
get_payment( )
raise_ExitBarrier( )