6. Car Rental Case Study in UML
6.1 Getting Started

- To follow: A medium sized case study for a car rental administration system
- Explanation of central UML diagrams and language features therein
- Used here: Use case, class, object, statechart, sequence, collaboration, and activity diagrams
- Development of the case study also demonstrates a typical development process
- Diagram order in this document does not reflect the order occurring in the development
Overview on the Used Development Process

(1) Start with a use case diagram giving an overview on the system, its boundaries and its main functionality

(2) Develop object diagrams and sequence diagrams as desired scenarios for the system structure and behavior

(3) Develop a class diagram and a statechart diagram for each class; start with a textual description of the statechart diagrams; continue with a more formal description for the statecharts
Overview on the Used Development Process

(4) Check whether the existing object diagrams and sequence diagrams are consistent with the class diagram and the statechart diagrams; if necessary, modify them.

(5) Develop more object and sequence diagrams.

(6) If an operations seems to become stable (no changes in the diagrams for the parts with that operation), start to develop activity diagrams for that operation; begin with a textual description and advance to a more formal one.

(7) Repeat steps (4)-(6) until an acceptable model is found.
6.2 Use Cases
A real world customer is outside the Car Rental System, a customer within that system.

**Use Case Diagram**

- **Car Rental System**
  - create customer
  - create car
  - book
  - cancel
  - pickUp
  - return
  - <<extend>>
    - <<extend>>
      - early return
      - late return

- **<<actor>>**
  - user
  - request
  - billing

- **<<extend>>**
  - billing clerk
  - PersistentStore

- **real world customer**
  - user
  - billing
  - request
Details for Use Case create customer

Use case name: create customer
Goal: to create a new customer
Precondition: the real world customer to be recorded is currently not represented
Postcondition: a new customer exists
Actors: user
Triggering event: a real world customer must be recorded
Description: -
Extensions: -
Alternatives: -
Details for Use Case create car

Use case name: create car
Goal: to create a new car
Precondition: the real world car to be recorded is currently not represented
Postcondition: a new car exists
Actors: user
Triggering event: a real world car must be recorded
Description: -
Extensions: -
Alternatives: -
Details for Use Case book

Use case name: book
Goal: to enter a car rental booking
Precondition: the booking details are plausible
Postcondition: a new booking exists; the booking is now an open booking
Actors: user
Triggering event: a real world customer requests a booking
Description: the real world customer wants to rent a real world car of a certain category; start day of the rental is the current day or a day after the current day; end day of the rental lies after the start day
Extensions: -
Alternatives: -
Details for Use Case cancel

Use case name: cancel
Goal: to prevent that a car must be picked up for a booking
Precondition: the booking to be canceled is present
Postcondition: the booking is marked as closed; no car will be picked up for this booking
Actors: user
Triggering event: a real world customer requests a cancelation of a booking
Description: -
Extensions: -
Alternatives: triggering event - the start day of a booking is passed and no car has been picked up by the customer for that booking
Details for Use Case pickUp

Use case name: pickUp
Goal: to deliver a car for a car rental
Precondition: a booking is present
Postcondition: a suitable car is marked as unavailable (a real world car is given to a real world customer); the booking becomes a current booking
Actors: user
Triggering event: a real world customer requests a pick up
Description: a suitable car must be found among the currently available cars; if none is present, a new car may be added (a new real world car is purchased)
Extensions: –
Alternatives: –
Details for Use Case return

Use case name: return
Goal: to return a car for a car rental
Precondition: a current booking exists and a car has been delivered
Postcondition: the booking becomes closed; the car becomes available (real world customer has returned real world car)
Actors: user
Triggering event: a real world customer requests a return on the end day of the booking
Description: -
Extensions:
  early return: a rented car is returned before the end date of the booking
  late return: a rented car is returned after the end date of the booking
Alternatives: -
6.3 Static Structure Diagrams
6.3 Static Structure Diagrams

Class Diagram

**Customer**
- Name : String
  - create(n:String):Customer
  - book(s:DateT,e:DateT,ct:CatET)
  - cancel(b:Booking)
  - pickUp(b:Booking):Car
  - return(cr:Car)

**Car**
- LicenceNo : String
  - Cat : CatET
  - create(l:String,ct:CatET):Car

**Booking**
- Start : DateT
- End : DateT
- Cat : CatET
- PickedUp : Boolean
  - create(cs:Customer,s:DateT,e:DateT,ct:CatET):Booking
  - searchAvailCar():Car
  - assignCar(cr:Car)
  - returnCar()

**Enumeration**
- CatET
  - E / economy
  - S / standard
  - L / luxus

**Structure**
- DateT
  - Day : Integer <<1..31>>
  - Month : Integer <<1..12>>
  - Year : Integer <<1900..2100>>

Attention! many "returns":
- Customer::return(cr:Car)
- Booking::return(cr:car)
- Booking::Return:dateT
- Booking::returnCar()
Allowed Object Diagram: 3 Classes, 3 Objects

Today="11/22"

**ada:Customer**
- Name="Ada"

**b:Booking**
- Start="11/20"
- End="11/23"
- Cat=E
- PickedUp=true
- Return=undef

**vw:Car**
- LicenceNo="ABC123"
- Cat=S

Registration

booking

Assignment

Customer

Booking

Assignment

Car
Allowed Object Diagram: 3 Classes, 6 Objects

- ada:Customer
  - Name="Ada"
  - Registration
  - booking

- b1:Booking
  - Start="11/20"
  - End="11/23"
  - Cat=E
  - PickedUp=true
  - Return=undef
  - Assignment
car

- vw:Car
  - LicenceNo="ABC123"
  - Cat=S

- b2:Booking
  - Start="11/16"
  - End="11/18"
  - Cat=L
  - PickedUp=true
  - Return="11/17"
  - Assignment
car

- bob:Customer
  - Name="Bob"
  - Registration
  - booking

- ford:Car
  - LicenceNo="DEF456"
  - Cat=E

Today="11/22"
Disallowed Object Diagram

1. Booking b1 without customer
2. Car vw has 2 bookings
3. Booking b2 has 2 cars
4. b2’s Cat is S, ford’s Cat is E

- Customer ada
  - Name: Ada
  - Registration

- Booking b1:
  - Start: 11/20
  - End: 11/23
  - Cat: E
  - PickedUp: false
  - Return: undefined

- Car vw:
  - LicenceNo: ABC123
  - Cat: S

- Booking b2:
  - Start: 11/22
  - End: 11/30
  - Cat: S
  - PickedUp: true
  - Return: undefined

- Car ford:
  - LicenceNo: DEF456
  - Cat: E

Today: 11/23
6.4 Statechart Diagrams
Main Idea of Customer Statechart

- noReturns
  - create()
    - book() → booked
  - cancel() → mustReturn
  - return() → pickUp()

- booked

- mustReturn

create(), book(), cancel(), return(), pickUp()
6.4 Statechart Diagrams

Customer Statechart with Textual Details

BOOK :
book(s,e,ct) [paramsOk] / createNewBooking

PICKUP :
pickUp(b) [bookingBelongsToCustomerAndIsDue] / findAvailableCarAndLinkBookingToFoundCar

create(n) / createNewCustomer

noReturns

return(cr) [onlyOneCurrentBookingForCustomer and existsOpenDueBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

return(cr) [moreThanOneCurrentBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

cancel(b) [bookingBelongsToCustomer and bookingOpenOrDue and moreThanOneBookingForCustomer] / recordCancellation

cancel(b) [bookingBelongsToCustomer and bookingOpenOrDue and onlyOneBookingForCustomer] / recordCancellation

return(cr) [onlyOneCurrentBookingForCustomer and existsOpenDueBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

return(cr) [onlyOneCurrentBookingForCustomer and noOpenDueBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

mustReturn

BOOK

booked

cancel(b) [bookingBelongsToCustomer and bookingOpenOrDue] / recordCancellation

PICKUP

return(cr) [onlyOneCurrentBookingForCustomer and existsOpenDueBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

return(cr) [moreThanOneCurrentBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

return(cr) [onlyOneCurrentBookingForCustomer and existsOpenDueBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

return(cr) [moreThanOneCurrentBookingForCustomer and carBelongsToCustomer and carUnavailableOrDue] / handleBooking

cancel(b) [bookingBelongsToCustomer and bookingOpenOrDue] / recordCancellation
Booking Statechart with Textual Details

- **open**
  - `create(cs,s,e,ct)` / `createNewBooking`
  - `assignCar(cr)` / `linkBookingAndCarAndRecordPickUp`
  - `when(todayIsTheBookingsStartDay)`
  - `when(todayIsAfterTheBookingStartDay)` / `recordCancelation`

- **due**
  - `customerCancels` / `recordCancelation`
  - `assignCar(cr)` / `linkBookingAndCarAndRecordPickUp`
  - `when(todayIsAfterTheBookingStartDay)` / `recordCancelation`

- **current**
  - `returnCar()` / `unlinkBookingAndCarAndRecordReturnDay`

- **closed**
  - `returnCar()` / `unlinkBookingAndCarAndRecordReturnDay`
Car Statechart with Textual Details

- **create(l,ct) / createNewCar**
- **available**
  - **carAssigned**
  - **carReturned**
- **unAvailable**
- **due**

*when(todayIsTheBookingsEndDay)*
Customer Statechart with Formal Details

**BOOK**
- (booking->select(open)->size>0 or booking->select(due)->size>0) and booking->select(current)->size=0
- create(n) / Name=n
- cancel(b) [CANOK and CAN2noReturns]

**PICKUP**
- return(cr) [RETOK and RET2booked] / RETACT
- BOOK
- mustReturn
- booking->select(open)->size=0 and booking->select(due)->size=0 and booking->select(current)->size=0
- BOOK
- cancel(b) [CANOK and CAN2booked]
- BOOK
- cancel(b) [CANOK and CAN2noReturns]

**RETOK**
- return(cr) [RETOK and RET2noReturns] / RETACT

**RET2noReturns**
- (booking->select(open)->size=0 and booking->select(due)->size=0) and booking->select(current)->size=1

**RET2booked**
- (booking->select(open)->size>0 or booking->select(due)->size>0) and booking->select(current)->size=1

**RET2mustReturn**
- booking->select(current)->size>1

**RETACT**
- cr.booking.returnCar()
Customer Statechart with Formal Details Using Junction Points

BOOK

BOOK: book(s,e,ct) [ Today<=s and s<= e ] / booking.create(self,s,e,ct)

PICKUP: pickUp(b) [ booking->includes(b) and b.due ] / result=b.searchAvailCar(); b.assignCar(result)

CANOK: booking->includes(b) and (b.open or b.due)

CAN2noReturns: booking->select(open or due)->size=1

CAN2booked: booking->select(open or due)->size>1

RETOK: booking.car->includes(cr) and (cr.unAvailable or cr.due)

RET2noReturns: ( booking->select(open)->size=0 and booking->select(due)->size=0 ) and booking->select(current)->size=1

RET2booked: ( booking->select(open)->size>0 or booking->select(due)->size>0 ) and booking->select(current)->size=1

RET2mustReturn: booking->select(current)->size>1

RETACT: cr.booking.returnCar()

noReturns

[ CAN2noReturns ]

create(n) / Name=n

[ CAN2booked ]

cancel(b) [ CANOK ]

[ RET2noReturns ]

[ RET2booked ]

return(cr) [ RETOK / RETACT ]

[ RET2noReturns ]

booking->select(open)->size=0 and booking->select(due)->size=0 and booking->select(current)->size=0

booking->select(current)->size>0

[ CAN2noReturns ]

[ CAN2booked ]

[ CANOK ]
6.4 Statechart Diagrams

Booking Statechart With Formal Details

```
create(cs,s,e,ct) / Start=s; End=e; Cat=ct; link(Registration,[cs,self])

open

closed

due

when(Today=Start)

customer.cancel(self) / PickedUp=false

when(Today>Start) /
PickedUp=false;
customer.cancel(self)

current

assignCar(cr) / 
link(Assignment,[self,cr]); 
PickedUp=true

returnCar() / unlink(Assignment,[self,car]); Return=Today
```
Car Statechart with Formal Details

create(l,ct) / LicenceNo=l; Cat=ct

available

booking.assignCar(self)

unAvailable

booking.returnCar()

booking.returnCar()

due

when(Today=booking.End)
Sequence Diagram for Booking: `Car.create; Booking.create`

Today = b.Start

- `create("Ada")`
- `create("ABC123", S)`
- `create(ada,"11/20", "11/23", E)`

Today = b.End

- `pickUp(b)`
- `searchAvailCar()`
- `assignCar(vw)`
- `returnCar()`
- `return(vw)`

States:
- `mustReturn`
- `unAvailable`
- `available`
- `due`
- `current`
- `closed`
- `open`
- `booked`
- `current`
- `closed`
- `booked`
- `open`
- `current`
Sequence Diagram for Booking:

Car.create; Booking.create (with objects)
6.5 Collaboration Diagrams
Collaboration Diagram for Booking: \texttt{Car.create; Booking.create}

1. create("Ada")
2. create("ABC123", S)
4. pickUp(b)
5. return(vw)

3.1 create(ada,"11/20","11/23",E)
4.1 searchAvailCar()
4.2 assignCar(vw)
5.1 returnCar()
Sequence Diagram for Booking: Booking.create; Car.create
Sequence Diagram for Booking:
Booking.create; Car.create (with objects)
Collaboration Diagram for Booking: Booking.create; Car.create

1. create("Ada")
3. pickUp(b)
4. return(vw)

2.1 create(ada,"11/20","11/23",E)
3.1 searchAvailCar()
3.3 assignCar(vw)
4.1 returnCar()
Sequence Diagram for Booking with Implicit Cancel

Today = b.Start
Today > b.Start

create("Ada")
cancel(b)

ada: Customer
noReturns

cancel(b)

b: Booking
open
due
closed
due
noReturns

Today = b.Start
Today > b.Start
Sequence Diagram for Booking with Explicit Cancel

:User
create("Ada") -> ada:Customer

noReturns


booked

open

cancel(b) -> noReturns

closed
Sequence Diagram for Booking with Early Return

- User
  - create("Ada")
  - create("ABC123", S)

- Customer
  - Today=b.Start
  - Today=b.End

- Booking
  - due
  - closed
  - booked
  - noReturns

- Car
  - available
  - searchAvailCar()
  - assignCar(vw)

- return(vw)

- returnCar()
Sequence Diagram for Booking with Late Return

Today=b.Start

create("Ada")
create("ABC123",S)
create(ada,"11/20","11/23",E)

pickUp(b)

searchAvailCar()

assignCar(vw)

return(vw)

returnCar()

Today=b.End

create("Ada")
create("ABC123",S)
create(ada,"11/20","11/23",E)

pickUp(b)

searchAvailCar()

assignCar(vw)

return(vw)

returnCar()
Sequence Diagram for Two Bookings: \texttt{b1.return; b2.assignCar}
Sequence Diagram for Two Bookings: \texttt{b2.assignCar; b1.return}
6.6 Closing
Activity Diagrams for `pickUp`: Textual Level and Operation Level

```java
Customer::pickUp(b):Car

result = b.searchAvailCar()

if result.available
    chooseAnAvailableCar
else
    purchaseANewCar

connectTheBookingToTheCar
indicateThatACarHasBeenPickedUp
returnTheCar

return(result)

Customer::pickUp(b):Car

result = b.searchAvailCar()

b.assignCar(result)
return(result)
```
Activity Diagrams for `pickUp`: Statement Level and Formal Level

Customer::pickUp(b):Car

- testAvailableCars
  - [carAvailable] result=someAvailableCar
    - link(Assignment,[b,result])
      - b.pickedUp=true
      - return(result)
  - [else] result=someNewCar

Customer::pickUp(b):Car

- Car.allInstances->exists(cr| cr.available and cr.Cat>=b.Cat)
  - result:=Car.allInstances->select(cr|cr.available and cr.Cat>=b.Cat)->any
    - result:=Car.create(newLicNo,b.Cat)
    - b.car:=b.car->including(result)
    - b.pickedUp:=true
    - return(result)
- [false]

Customer::pickUp(b):Car

- Car.allInstances->exists(cr| cr.available and cr.Cat>=b.Cat)
  - result:=Car.allInstances->select(cr|cr.available and cr.Cat>=b.Cat)->any
    - result:=Car.create(newLicNo,b.Cat)
    - b.car:=b.car->including(result)
    - b.pickedUp:=true
    - return(result)
- [true]
Developing a Consistent Model

- develop class diagrams
- develop statechart diagrams
- develop object diagrams
- develop sequence diagrams

[diagrams ok]
[diagrams inconsistent or incomplete]
[more diagrams needed]
Details for Diagram Consistency

From class diagram to object diagrams
- for each class there is at least one object diagram with an object of that class
- for each attribute and role name there is at least one object diagram with an object with that attribute and role name

From class diagram to sequence diagrams
- for each class there is at least one sequence diagram with an object of that class
- for each operation there is at least one sequence diagram with that operation as a message

From class diagram to statechart diagrams
- for each attribute there is at least one operation that modifies that attribute in some statechart
- for each operation there is at least one statechart where that operation occurs as a call event or as a call action

From statechart diagrams to class diagram
- each call event refers to an operation in a class
- each attribute and role name occurring in a guard refers to an attribute in a class and a role name of a class
- each call action, assignment action, and link/unlink action refers to an operation, attribute, and role name in a class

From object diagrams to class diagram
- each class of an object occurs as a class
- each attribute and role name refers to an attribute in a class and a role name of a class

From sequence diagram to class diagram
- each object refers to a class
- each message refers to an operation in a class

From sequence diagram to statechart diagram
- each state refers to a state in the statechart diagram
- each message sequence is allowed by at least one statechart event and action order
- each state sequence is allowed by at least one state sequence order

From statechart diagram to the sequence diagram
- each call action completely occurs in the sequence diagram (all sub-actions from the call action occur)