6. Car Rental Case Study in UML

- 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
6. Car Rental - 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

(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 operation 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
A **real world customer** is outside the Car Rental System, a **customer** within that system.
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: -
6. Car Rental - 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: -
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: -
6. Car Rental - 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
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: -
6. Car Rental - 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. Car Rental - 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)

1 customer
Registration

0..* booking

Attention! many "returns":
Customer::return(cr:car)
Booking::Return:dateT
Booking::returnCar()

Car

| LicenceNo : String
| Cat : CatET |

create(l:Character ct:CatET):Car

Car

| car : Car |

car : Set(Car)

Assignment

0..1 car

Car

0..1 booking

Only correct dates

Booking

| Start : DateT
| End : DateT
| Cat : CatET |

create(cs:Customer,s:DateT,e:DateT,ct:CatET):Booking
searchAvailCar():Car
assignCar(cr:Car)
returnCar()

create

searchAvailCar():Car
post Cat<=result.Cat and result.available

Attention! many "returns":
Customer::return(cr:car)
Booking::Return:dateT
Booking::returnCar()
6. Car Rental - 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

Assignment
car

Registration
customer

Assignment
booking

Registration
booking
6. Car Rental - Allowed Object Diagram: 3 Classes, 6 Objects

- ada:Customer
  - Name="Ada"

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

- bob:Customer
  - Name="Bob"

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

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

- ford:Car
  - LicenceNo="DEF456"
  - Cat=E
6. Car Rental - Disallowed Object Diagram

<table>
<thead>
<tr>
<th>Customer</th>
<th>Booking b1</th>
<th>Car vw</th>
<th>Booking b2</th>
<th>Car ford</th>
</tr>
</thead>
</table>

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

PICKUP:
pickUp(b) [bookingBelongsToCustomerAndIsDue] / findAvailableCarAndLinkBookingToFoundCar

noReturns
create(n) / createNewCustomer

returned(cr) [onlyOneCurrentBookingForCustomer and noOpenDueBookingForCustomer 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 and moreThanOneBookingForCustomer] / recordCancelation

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

cancel(b) [bookingBelongsToCustomer and bookingOpenOrDue] / handleBooking
6. Car Rental - Booking Statechart with Textual Details

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

- **due**
  - when(todayIsAfterTheBookingStartDay) / recordCancelation

- **closed**

- **current**
6. Car Rental - Car Statechart with Textual Details

available

create(l,ct) / createNewCar

carReturned

carReturned

carAssigned

unAvailable

when(todayIsTheBookingsEndDay)

due
6. Car Rental - Customer Statechart with Formal Details

- **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()
6. Car Rental - Customer Statechart with Formal Details Using Junction Points

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()

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

create(n) / Name=n

noReturns

[ CAN2noReturns ]

[ CAN2booked ]

[ RET2noReturns ]

[ RET2booked ]

[ RET2mustReturn ]

mustReturn

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

[ CANOK ]

[ RETOK ] / RETACT

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

[ CANOK ]

[ RETACT ]
6. Car Rental - Booking Statechart With Formal details

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

- **open**
  - when(Today=Start)
  - assignCar(cr) / 
    link(Assignment,[self,cr]);
  - PickedUp=true

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

- **closed**
  - returnCar() / unlink(Assignment,[self,car]);
  - Return=Today

- **current**
6. Car Rental - Car Statechart with Formal Details

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

available

booking.assignCar(self)

unAvailable

booking.returnCar()

when(Today=booking.End)

booking.returnCar()

due

booking.returnCar()
```

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

available

booking.assignCar(self)

unAvailable

booking.returnCar()

when(Today=booking.End)

booking.returnCar()
6. Car Rental - Sequence Diagram for Booking: Car.create; Booking.create

- User
  - create("Ada") → ada:Customer
    - noReturns
  - create("ABC123",S) → vw:Car
    - available
    - booked
  - pickUp(b) → b:Booking
    - open
    - due
  - searchAvailCar() → vw
  - assignCar(vw) → unAvailable
    - current
  - mustReturn
  - return(vw) → returnCar()
    - noReturns
    - available
    - closed

Today = b.Start
Today = b.End
6. Car Rental - Sequence Diagram for Booking:
   `Car.create; Booking.create` (with objects)
6. Car Rental - Collaboration Diagram for Booking: Car.create; Booking.create

- ada:Customer {new}
- vw:Car {new}
- b:Booking {new}
- :User

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()
6. Car Rental - Sequence Diagram for Booking: Booking.create; Car.create

Today = b.Start
create(Ada) → ada:Customer
     noReturns
     booked
     open
     due
pickUp(b) → searchAvailCar()
create("ABC123",S) → create(abc123,S) → create(abc123,S)
     available
     unAvailable
     current
    return(vw)
       returnCar()
       due
       available
       closed

Today = b.End

User

ada:Customer
vw:Car
b:Booking
6. Car Rental - Sequence Diagram for Booking:
Booking.create; Car.create (with objects)
6. Car Rental - 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()
6. Car Rental - Sequence Diagram for Booking with Implicit Cancel

Today=b.Start

Today>b.Start
create("Ada")

create(ada,"11/20","11/23",E)

cancel(b)

ada:Customer

noReturns
due
closed


create(ada,"11/20","11/23",E)

b:Booking

booked

open
due

cancel(b)

noReturns

open
due

closed
6. Car Rental - Sequence Diagram for Booking with Explicit Cancel

Today
create("Ada")
create(ada,"11/20","11/23",E)
cancel(b)
ada:Customer
b:Booking
closed
noReturns
open
:User
booked
noReturns
Today<b.Start
create("Ada")
create(ada,"11/20","11/23",E)
cancel(b)
ada:Customer
b:Booking
closed
noReturns
open
:User
booked
noReturns
6. Car Rental - Sequence Diagram for Booking with Early Return

:User

create("Ada") \rightarrow ada:Customer

create("ABC123",S) \rightarrow vw:Car


pickUp(b) \rightarrow booked

searchAvailCar() \rightarrow vw

assignCar(vw) \rightarrow available

return(vw) \rightarrow unAvailable

mustReturn

returnCar() \rightarrow available

noReturns

Today=b.Start

Today=b.End

b:Booking

open
due

available

current

unAvailable

closed

30
6. Car Rental - Sequence Diagram for Booking with Late Return

Today=b.Start

create("Ada")
create("ABC123",S)
pickUp(b)
searchAvailCar()
assignCar(vw)
return(vw)

Today=b.End

returnCar()
return(vw)
nodReturns

vw:Car
:User
ada:Customer
available
unAvailable
due

mustReturn

due
current

closed
available
noReturns

open
due
closed
booked
noReturns
due
closed
booked
noReturns

Today=b.Start
Today=b.End
6. Car Rental - Sequence Diagram for Two Bookings: b1.return; b2.assignCar

Today=b1.Start
Today=b1.End
Today=b2.Start
assignCar(vw)
return(vw)
assignCar(ford)
current
closed
open
current
... ford:Car
vwb1:Booking
b2:Booking
ada:Customer
... available
unAvailable
due
open
due
unAvailable
unAvailable
available
closed
available
mustReturn
return(vw)
assignCar(ford)
6. Car Rental - Sequence Diagram for Two Bookings: b2.assignCar; b1.return

Today=b1.Start
Today=b1.End
Today=b2.Start

assignCar(vw)
return(vw)
assignCar(ford)
6. Car Rental - Activity Diagrams for `pickUp`: Textual Level and Operation Level

```
result = b.searchAvailCar()
b.assignCar(result)
testAvailableCars
chooseAnAvailableCar
connectTheBookingToTheCar
indicateThatACarHasBeenPickedUp
returnTheCar
```

```
result = b.searchAvailCar()
b.assignCar(result)
return(result)
```
6. Car Rental - Activity Diagrams for \texttt{pickUp}: Statement Level and Formal Level

\begin{align*}
\text{Customer::pickUp(b):Car} & \\
\text{testAvailableCars} & \\
\text{[carAvailable]} & \text{[else]} \\
\text{result=someAvailableCar} & \text{result=someNewCar} \\
\text{link(Assignment,[b,result])} & \\
\text{b.pickedUp=true} & \\
\text{return(result)} & \\
\bullet & \\
\end{align*}

\begin{align*}
\text{result:=Car.allInstances->exists(cr| cr.available and cr.Cat>=b.Cat)->any} & \\
\text{[true]} & \text{[false]} \\
\text{result:=Car.create(newLicNo,b.Cat)} & \\
\text{b.car:=b.car->including(result)} & \\
\text{b.pickedUp:=true} & \\
\text{return(result)} & \\
\bullet & \\
\end{align*}
6. Car Rental - Developing a Consistent Model

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

[diagrams inconsistent or incomplete]

[more diagrams needed]

[diagrams ok]

- modify object diagram
- modify class diagram
- modify statechart diagram
- modify sequence diagram

- develop more sequence diagrams
- develop more object diagrams
6. Car Rental - 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 as 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)