Class Project

Second Stage

Second Stage (1)

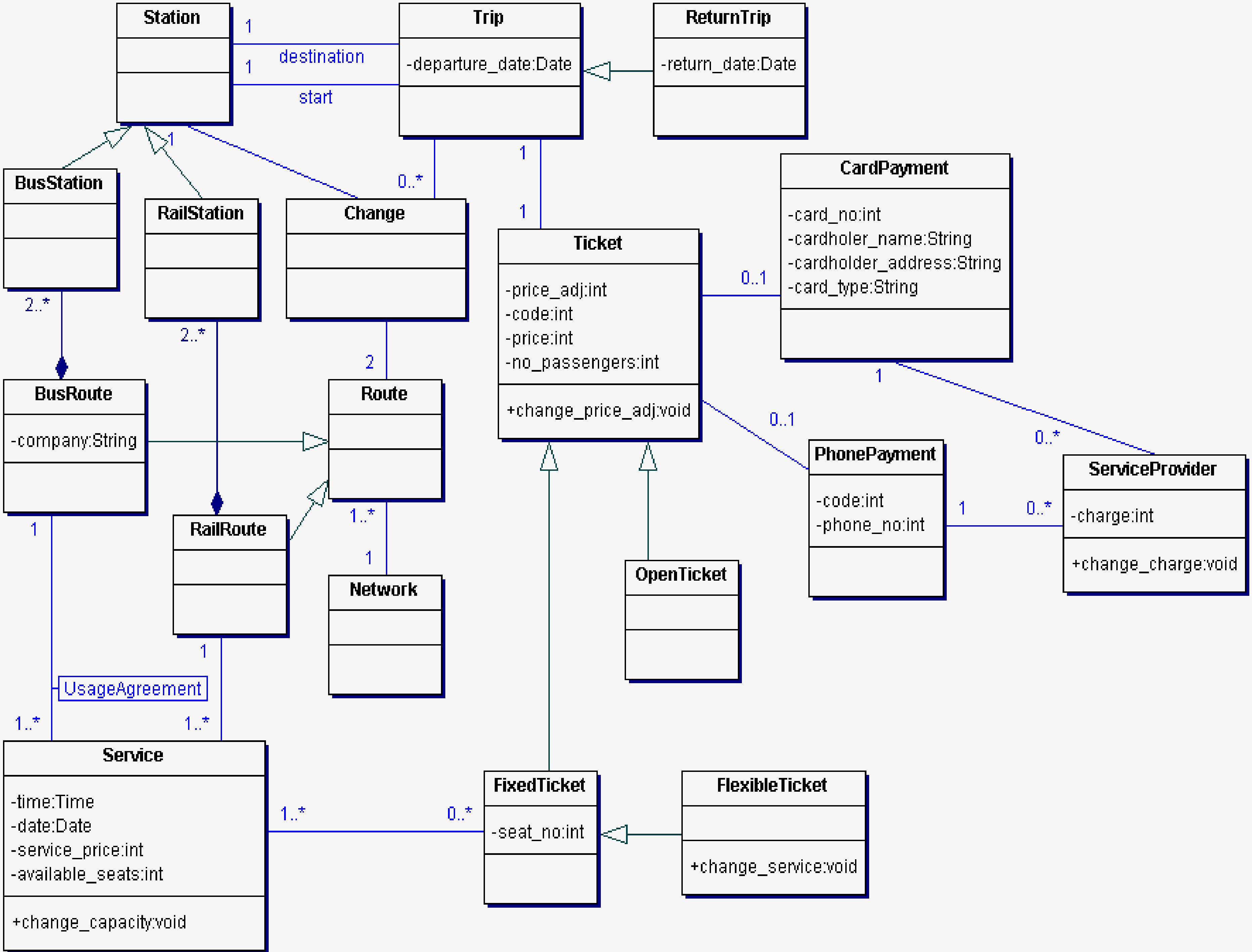
- Use case diagram
  - Customer/passenger functionality: purchase ticket, change ticket (provide trip details, pay ticket, issue ticket)
  - Check ticket validity
  - System management: view ticket sales, view revenue information, view service utilisation, update pricing information, update bus/rail routes, change commercial collaboration terms, change service capacity
  - Opportunities for generalisation
    - Actors: Tickets checker (Electronic gate, Ticket controller, Bus company staff)
    - Use cases: Pay ticket (Pay ticket with card, Pay ticket with phone), Issue ticket (Issue electronic ticket, Issue paper ticket)
  - The main issue is use case relationships!
    - What do they mean? When do we use them? How they are represented?
    - Important to remember that too many relationships complicate unnecessarily the diagrams!
  - Use cases represent actions/activities/functionality the system provides and they should be named in way that indicates this!
  - Balance of information provided is important!

Second Stage (2)

- Use case descriptions
  - Overall the best part of the system
  - The main problems are on the extended descriptions
    - No justification of the chosen use cases for extended description
    - Poor choice of use cases for extended description
    - Inconsistencies with the use case diagram
    - The template: name, actors, brief description, precondition, main flow, alternative flow, exceptional flow (optional), postconditions
      - All flows in actor action – system response format with numbers to indicate the sequencing

Second Stage (3)

- Class diagram
  - Quite good in general
  - Main problems
    - No documentation of design assumptions (e.g. why are certain attributes repeated in a subclass, or where ticket sales information is captured)
    - Some dubious classes and associations
    - Strange multiplicities


Station
Trip
-departure_date:Date
ReturnTrip
-return_date:Date
1
destination
1
start
BusStation
RailStation
Change
1
0..*
1
1
Ticket
-price_adj:int
-code:int
-price:int
-no_passengers:int
+change_price_adj:void
CardPayment
-card_no:int
-cardholer_name:String
-cardholder_address:String
-card_type:String
0..1
1
0..*
2..*
2..*
2
BusRoute
-company:String
Route
0..1
PhonePayment
-code:int
-phone_no:int
ServiceProvider
-charge:int
+change_charge:void
1
0..*
RailRoute
1..*
1
Network
1
OpenTicket
1
UsageAgreement
1..*
1..*
Service
-time:Time
-date:Date
-service_price:int
-available_seats:int
+change_capacity:void
1..*
0..*
FixedTicket
-seat_no:int
FlexibleTicket
+change_service:void

Second Stage (5)

- Activity diagrams
  - The main issue is inconsistencies with the use case descriptions!
  - Numerous examples of very poor understanding
    - Decision point with multiple exits labelled the same!
    - Decision point with all transitions leading to the same activity!!
    - Use of synchronisation bars to improve the look of the diagram!!!
    - Nonsensical activities (e.g. flexible ticket, fixed ticket etc.)
    - Loosing the system boundary

Second Stage (6)

- Sequence diagrams
  - Quite poor in general
  - The main problem is again consistency
    - Steps of the use case missing!
    - Operations that do not appear in the class diagram!
    - Classes that do not appear in the class diagram!!
  - Object lifelines
- Statechart
  - Quite poor in general
  - The main problem is that most statecharts are in fact activity diagrams!!
    - Strange classes are often introduced and then model with statecharts

Second Stage (7)

- Overall performance – quite good!
  - Marks were more spread in general
  - Best part the use case descriptions – (avg 70%)
  - Worst parts sequence diagrams and statecharts – (avg 50%)
  - 35% – 83% avg 60%