Train Depot Answers (1)

1. Change the load of a car (as a result of loading or unloading it). The new weight is a parameter of the operation.

   Operation: Depot :: changeLoad (target: Car, newLoad: Positive);

   Pre: newLoad ≤ target.maxLoad;

   Post: target.currentLoad = newLoad;

- What if the traction strength of the engines is exceeded?
  - Strengthen the precondition or
  - Send an error message

Train Depot Answers (2)

2. Add an (existing) car to a train.

Operation: Depot :: addCar(unit: Car, to: Train);
Pre: to.participant→size() < 25 &
to.totalTraction() ≥ to.totalWeight() + to.totalLoad() +
unit.weight + unit.currentLoad;
Post: to.participant→includes(unit);

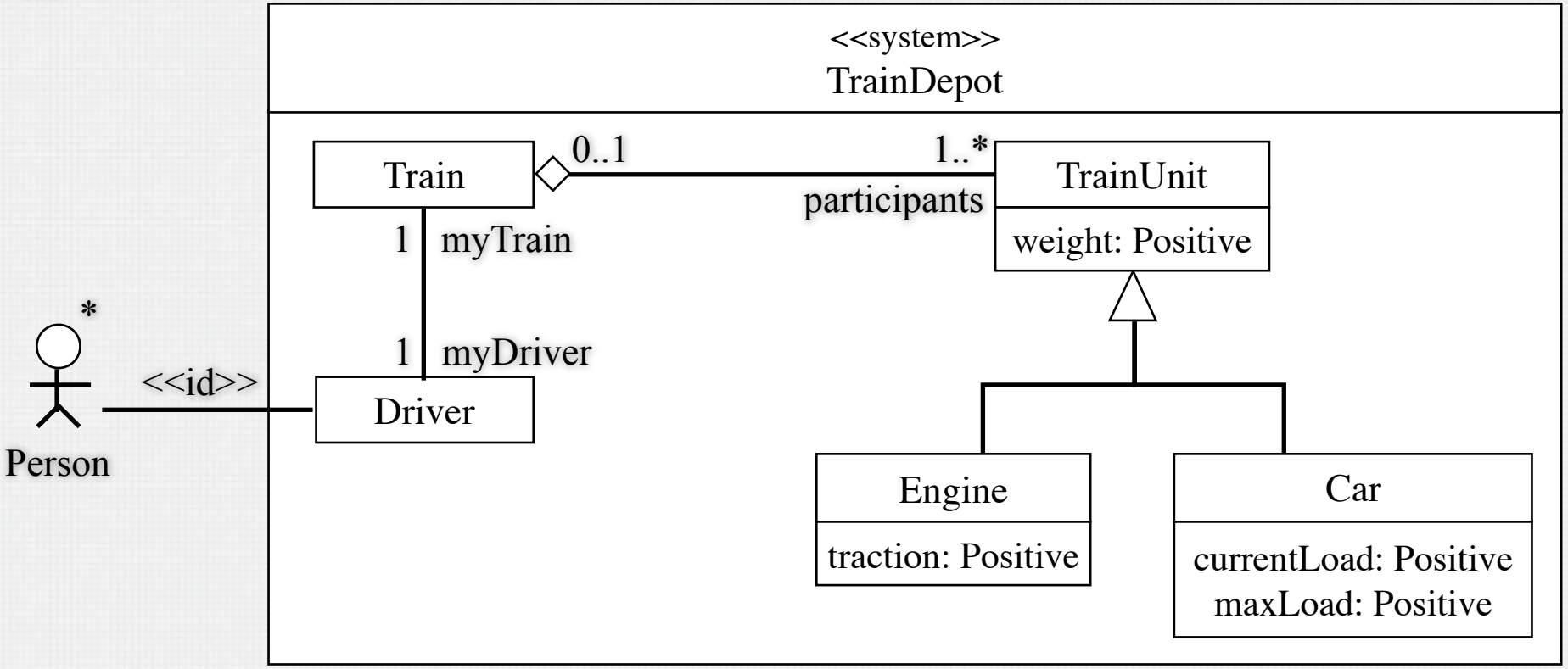
Train Depot Answers (3)

3. Transfer one train unit from one train to another one.

Operation: Depot::transfer (from: Train, unit: TrainUnit, to:Train);
Pre:from.participant→includes(unit) &
  if unit.oclIsTypeOf(Engine) then
  from.participant→select(u | u.oclIsType(Engine))→size() > 1
  endif & to.participant→size() < 25 &
  to.totalTraction() ≥ to.totalWeight() + to.totalLoad() + unit.weight +
    (if unit.oclIsTypeOf(Car) then unit.currentLoad else 0 endif);
Post: from.participant→excludes(unit) &
  to.participant→includes(unit);

Train Depot Answers (4)

4. Compute the total load weight of a train and communicate it to the driver.
   - We must add the concept of a driver to the Concept Model and connect it to the "real" driver actor with an <<id>> association



Train Depot Answers (5)

4. Compute the total load weight of a train and communicate it to the driver.

message type Weight(load: Positive);

Operation: Depot::computeLoad (target: Train);
Messages: Person::{Weight;};
Pre: **true**;
Post: target.driver.person^weight(target.participant→**select**(u | u.**oclIsTypeOf**(Car))→**collect**(currentLoad)→**sum**());

atFloor Answer

Operation: Elevator::atFloor(f: Floor);
Description: The cabin has reached a certain floor. It may either continue, or stop, depending on its destination and the requests for floors.
Scope: Floor; Cabin; Request
Aliases: needToStop: Boolean =

```
f.reqsForFloor→select(r | r.dir = Direction::any)→notEmpty() or
if self.cabin.mvnt = Movement::up then
  f.reqsForFloor→select(r | r.dir = Direction::up)→notEmpty()
else
  f.reqsForFloor→select(r | r.dir = Direction::down)→notEmpty()
endif
```

Messages: Motor::{Stop};
Pre: (**self**.cabin.mvnt = Movement::up **or** self.cabin.mvnt = Movement::down) **and** **self**.cabin.ds = DoorState::closed;
Post:

```
self.cabin.currentFloor = f &
if needToStop then
  self.cabin.motor^stop() &
  if self.cabin.mvnt@pre = Movement::goingUp then
    self.cabin.mvnt = Movement::stoppingUp
  else self.cabin.mvnt = Movement::stoppingDown
  endif
endif
```

Stopped Answer

Operation: Elevator::stopped;
Description: The cabin has stopped at a floor where there was a pending request. The system needs to open the door and inform each request source that the request has been serviced.
Scope: Floor; Cabin; Request
Aliases: servicedReqs: Set(Request) =
if self.cabin.mvnt@**pre** = Movement::stoppingUp **then**
self.cabin.currentFloor.reqsForFloor@**pre**→**select**(r | r.dir <> Direction::down)
else self.cabin.currentFloor.reqsForFloor@**pre**→select(r | r.dir <> Direction::up);
endif
Messages: Door::{Open}; ReqSource::{ServicedRequest};
Pre: **self**.cabin.mvnt = Movement::stopping **and self**.cabin.ds = DoorState::closed;
Post: **self**.cabin.door^open() &
self.cabin.ds = DoorState::open &
self.cabin.mvnt = Movement::stopped &
servicedReqs→**forAll**(r | r.reqSource^servicedRequest()) &
self.request→**excludesAll**(servicedReqs)

isClosed Answer (1)

Operation:Elevator::isClosed;
Description: The cabin door is now closed. The elevator now services the next pending request.
Scope: Floor; Cabin; Request
Aliases: workToDo : Boolean = **self**.request@**pre**→**notEmpty**();
goUp : Boolean = **self**.cabin.currentRequest.targetFloor.num
> **self**.cabin.currentFloor.num;
goDown : Boolean = **self**.cabin.currentRequest.targetFloor.num
< **self**.cabin.currentFloor.num;
switchDown : Boolean = **self**.cabin.currentRequest.targetFloor.num =
self.requests.targetFloor.num→**max**();
switchUp : Boolean = **self**.cabin.currentRequest.targetFloor.num =
self.requests.targetFloor.num→**min**();
Messages: Motor::{Up, Down};
Pre: self.cabin.mvnt = Movement::stopped **and self**.cabin.ds = DoorState::open;

isClosed Answer (2)

```
Post:  self.cabin.ds = DoorState::closed &
           if workToDo then
             if goUp then
               self.cabin.mvnt = Movement::goingUp &
               self.cabin.motor^up()
             elsif goDown then
               self.cabin.mvnt = Movement::goingDown &
               self.cabin.motor^down()
         elsif switchDown then
               self.cabin.currentRequest = self.requests→any(r | r.targetFloor.num =
               self.requests.targetFloor.num→min()) &
               self.cabin.mvnt = Movement::goingDown &
               self.cabin.motor^down()
         elsif switchUp then
               self.cabin.currentRequest = self.requests→any(r | r.targetFloor.num =
               self.requests.targetFloor.num→max()) &
               self.cabin.mvnt = Movement::goingUp &
               self.cabin.motor^up()
         endif
   endif
```

floorRequest Answer (1)

Operation: Elevator::floorRequest(f : Floor; dir: Direction);
Description: The elevator system receives an external or internal request.
Scope: Floor; Cabin; Request
Aliases: HigherRequest : Boolean = (**self**.cabin.mvnt@**pre** = Movement::goingUp **and**
f.num > **self**.currentRequest@**pre**.targetFloor.num);
LowerRequest : Boolean = (**self**.cabin.mvnt@**pre** = Movement::goingDown **and**
f.num < **self**.currentRequest@**pre**.targetFloor.num);
New: r : Request;
Messages: Motor::{Up, Down};
Pre: **true**;

floorRequest Answer (2)

```
Post:   if self.requests@pre→isEmpty() and
          self.cabin.currentFloor = f then
          if self.cabin.ds@pre = DoorState::closed then
            self.cabin.door^open() &
            self.cabin.ds = DoorState::open
          endif &
          sender^servicedRequest()
        else r.oclIsNew() & r.targetFloor = f & r.dir = dir & r.reqSource = sender &
          if self.requests@pre→isEmpty() and
            self.cabin.ds@pre = DoorState::closed then
            if f.num > self.cabin.currentFloor.num then
              self.cabin.motor^up() & self.cabin.mvnt = Movement::goingUp
            else self.cabin.motor^down() &
              self.cabin.mvnt = Movement::goingDown
            endif &
            self.cabin.currentRequest = r
          elsif HigherRequest or LowerRequest then
            self.cabin.currentRequest = r
          endif
        endif
```