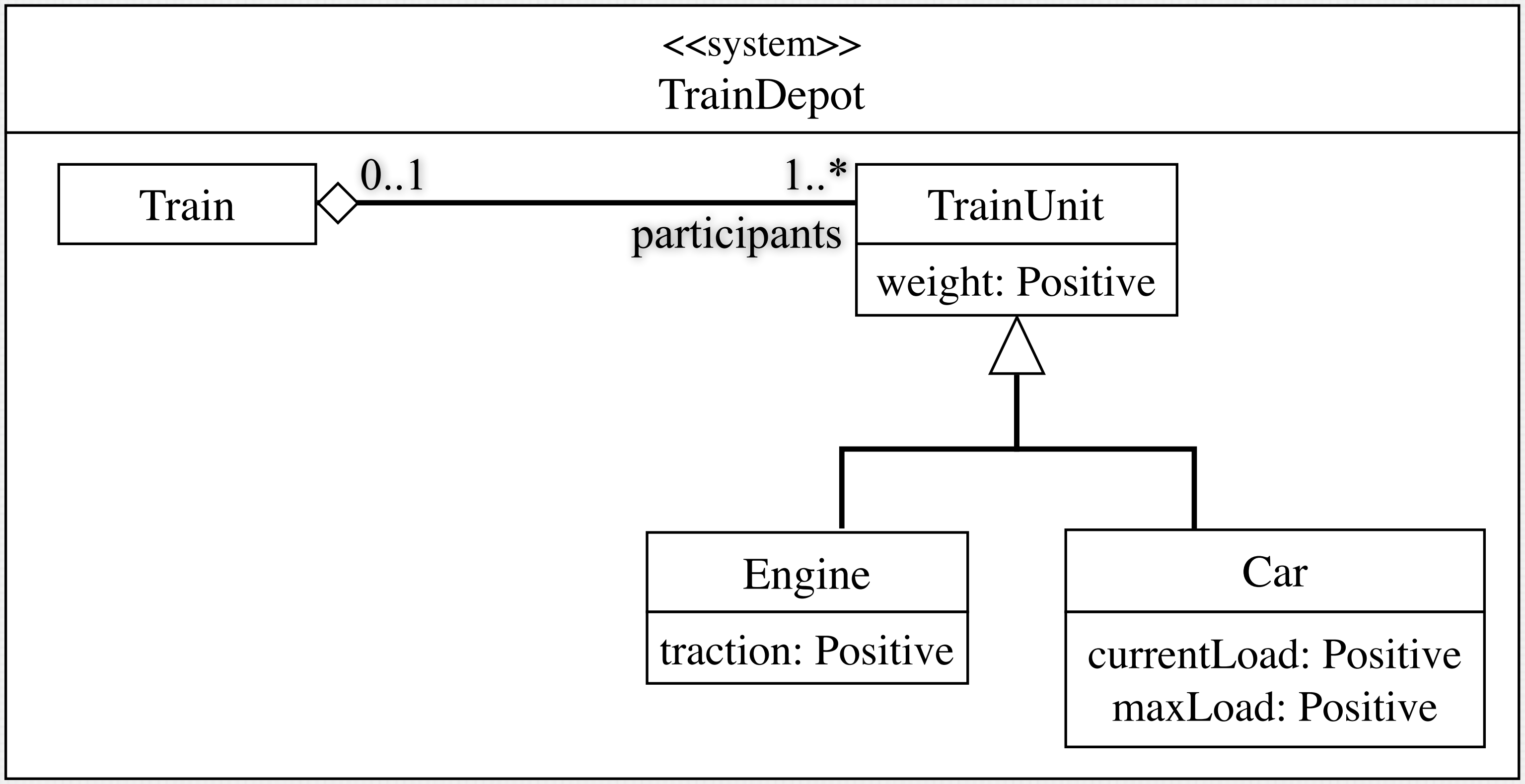
Train Depot Continued



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);
   Scope: Car
   Pre: The new load must be smaller or equal than the maximum load allowed for a car;
   Post: The operation updates the currentLoad value of the target car to the new load;

- 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);

Scope: Car, Train;

Pre: The traction strength of the engines in the train must be able to pull the additional weight of the car to be added;

Post: The operation associates the car unit with the train to (i.e., after the operation executed, unit is one of the participants of a train)

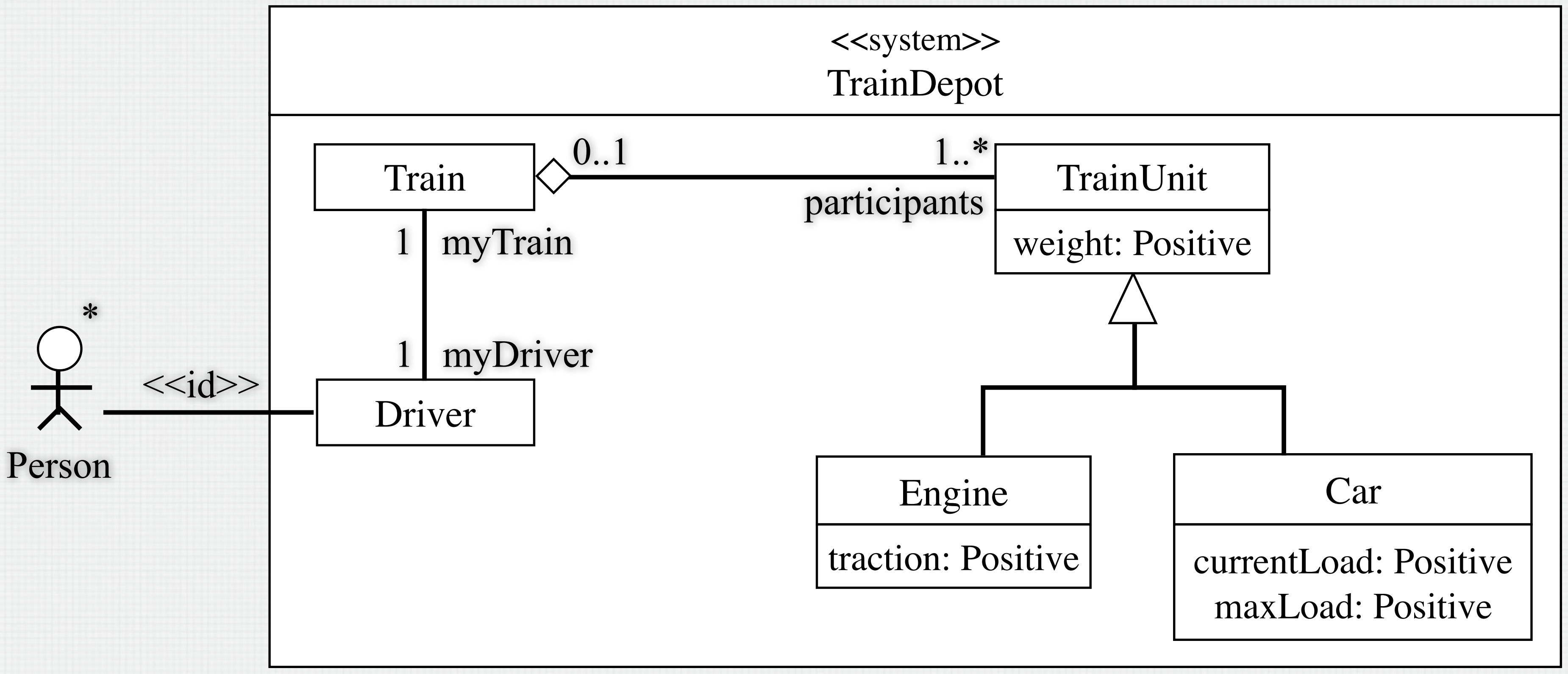
Train Depot Answers (3)

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

Operation: Depot::transfer (from: Train, unit: TrainUnit, to:Train);
Scope: Train, TrainUnit;
Pre: The total traction strength of the engines remaining in train from should be strong enough to pull the train. Likewise, the total traction strength of the engines of the train to must be strong enough to be able to pull the additional train unit unit.
Post: The operation removes the train unit *unit* from the train *from* and adds it to the train *to* (i.e., it deassociates unit from *from* and associates it with *to*).

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);
   Scope: Train, TrainUnit;
   Messages: Person::{Weight;};
   Post: The operation sums the weight of all the train units of the train target and the weight of their load (in case of cars) and sends an output message containing this information to the driver of the train;

atFloor Answer

Operation: Elevator::atFloor(f: Floor);

Scope: Floor; Cabin; Request

Messages: Motor::{Stop};

Pre: The cabin is currently moving up or down and the door is closed.

Post: The cabin state is updated to reflect the fact that it is now at floor f. If there are pending internal requests for this floor, or external requests for the same direction that the cabin is currently moving in, then the motor is instructed to stop by sending the stop output message.

Stopped Answer

Operation: Elevator::stopped;

Scope: Floor; Cabin; Request

Messages: Door::{Open}; ReqSource::{ServicedRequest};

Pre: The cabin state is currently stopping (up or down) and the door is closed;

Post: The cabin has stopped at a floor where there was a pending request. The cabin state needs to be updated to reflect that. The system needs to open the door and inform each request source of the floor that the request has been serviced by sending the appropriate output message. The serviced requests need to be deleted.

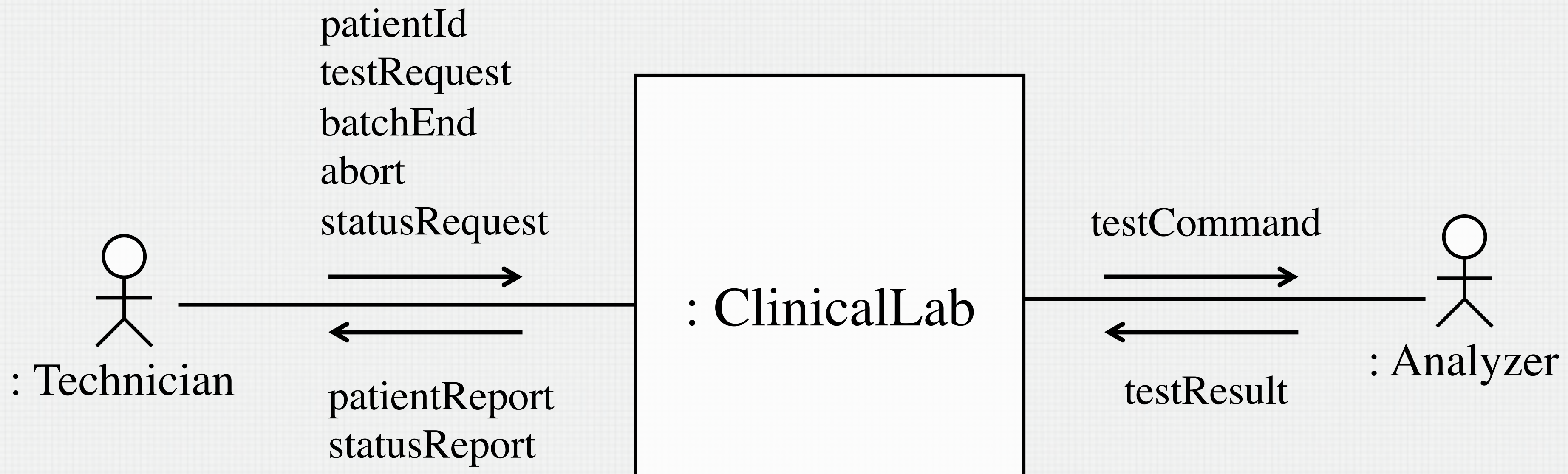
isClosed Answer (1)

Operation: Elevator::isClosed;
Scope: Floor; Cabin; Request
Messages: Motor::{Up, Down};
Pre: The motor is stopped and the door is open;
Post: The cabin door is now closed. The door state must be updated to reflect that. The elevator continues servicing the current request. If there are no requests to be serviced, nothing happens. If the current request is for a higher floor, the motor is asked to go up. If the current request floor was reached and there is no higher existing request, then the elevator needs to switch direction and start moving down by asking the motor. If the current request is for a lower floor, the motor is asked to go down. If the current request floor was reached and there is no lower existing request, then the elevator needs to switch direction and start moving up by asking the motor.

floorRequest Answer (1)

Operation: Elevator::floorRequest(f : Floor; dir: Direction);
Scope: Floor; Cabin; Request
Messages: Motor::{Up, Down}, Door::{Open};
New: newRequest: Request;
Post: The elevator system receives an external or internal request. If the request is for the floor the cabin is already on then the door is asked to open (if it isn't already). Otherwise a new request is created. If there were no other requests, the new request becomes the current request and, if the door is closed, the motor is asked to move in the appropriate direction. Otherwise (i.e. the elevator was serving some other request), a check is performed to see if the new request is higher than the current request and the cabin is moving up, or if the new request is lower than the current request and the cabin is moving down. If any of these conditions are met, the new request replaces the current request.

Clinical Lab System Answer (1)



Clinical Lab System Answers (2)

Message Declarations

type Test **is enum** {… names of lab tests on blood, urine, and swab...};
type State **is enum** {finished, ongoing};
type Measure **is TupleType** {kind: Test, value: Real};

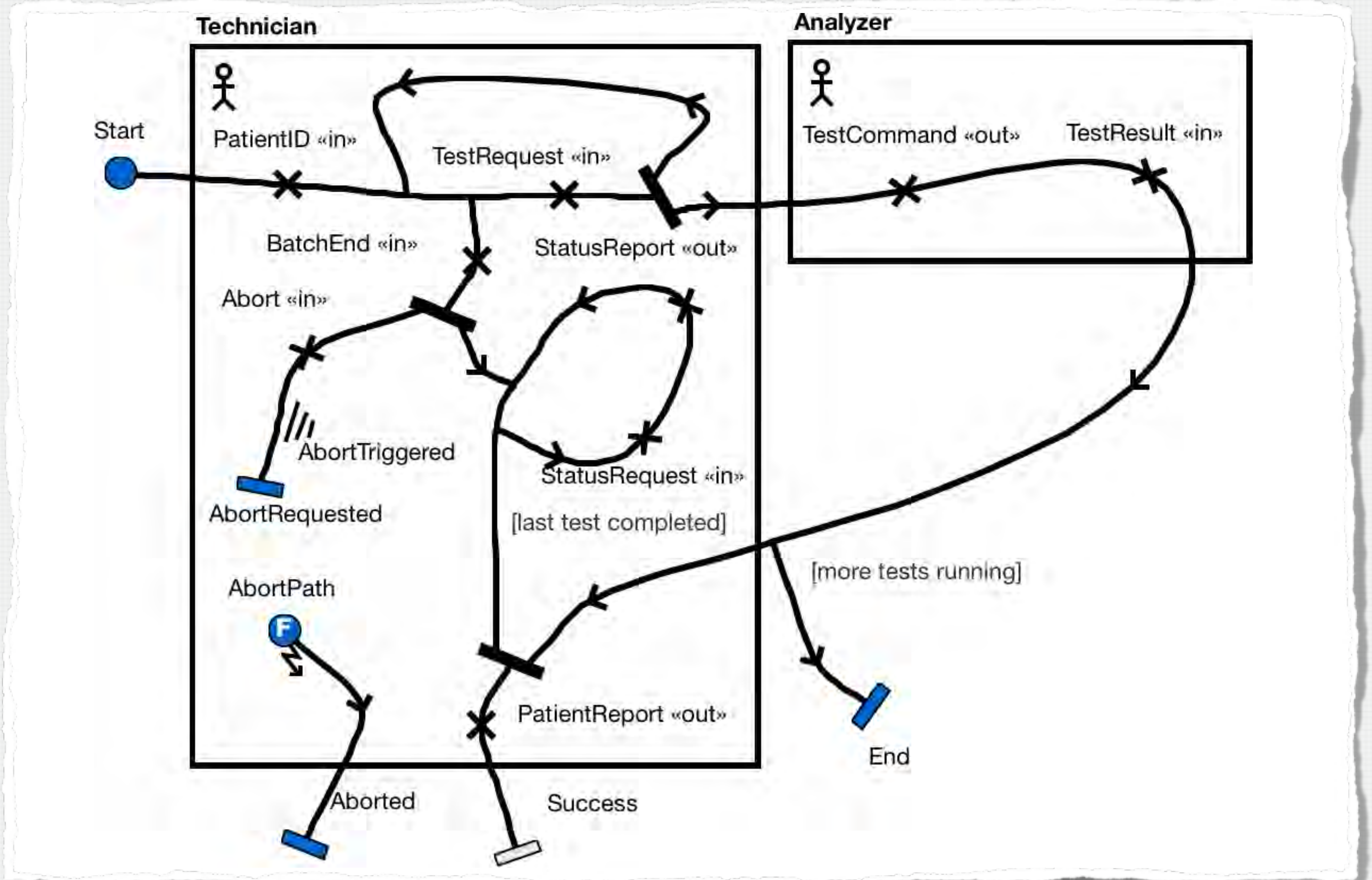
Input

PatientId (id: Integer, name: String);
TestRequest (id: Integer, kind: Test);
BatchEnd (id: Integer);
Abort (id: Integer);
StatusRequest (id: Integer);
TestResult (id: Integer, result: Measure);

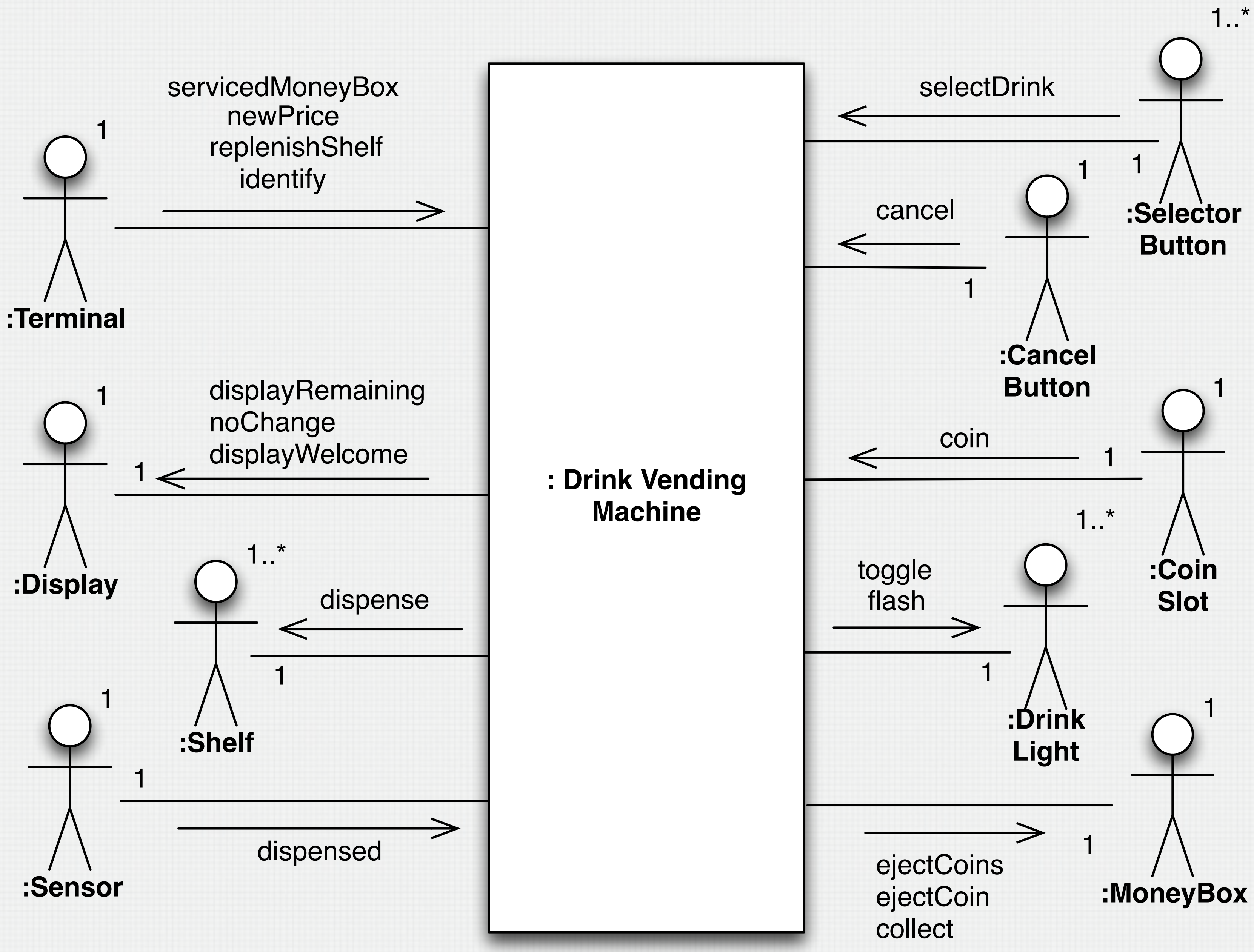
Output

TestCommand (id: Integer, kind: Test);
PatientReport (id: Integer, results: Collection (Measure));
StatusReport (id: Integer, status: State).

Clinical Lab System PM



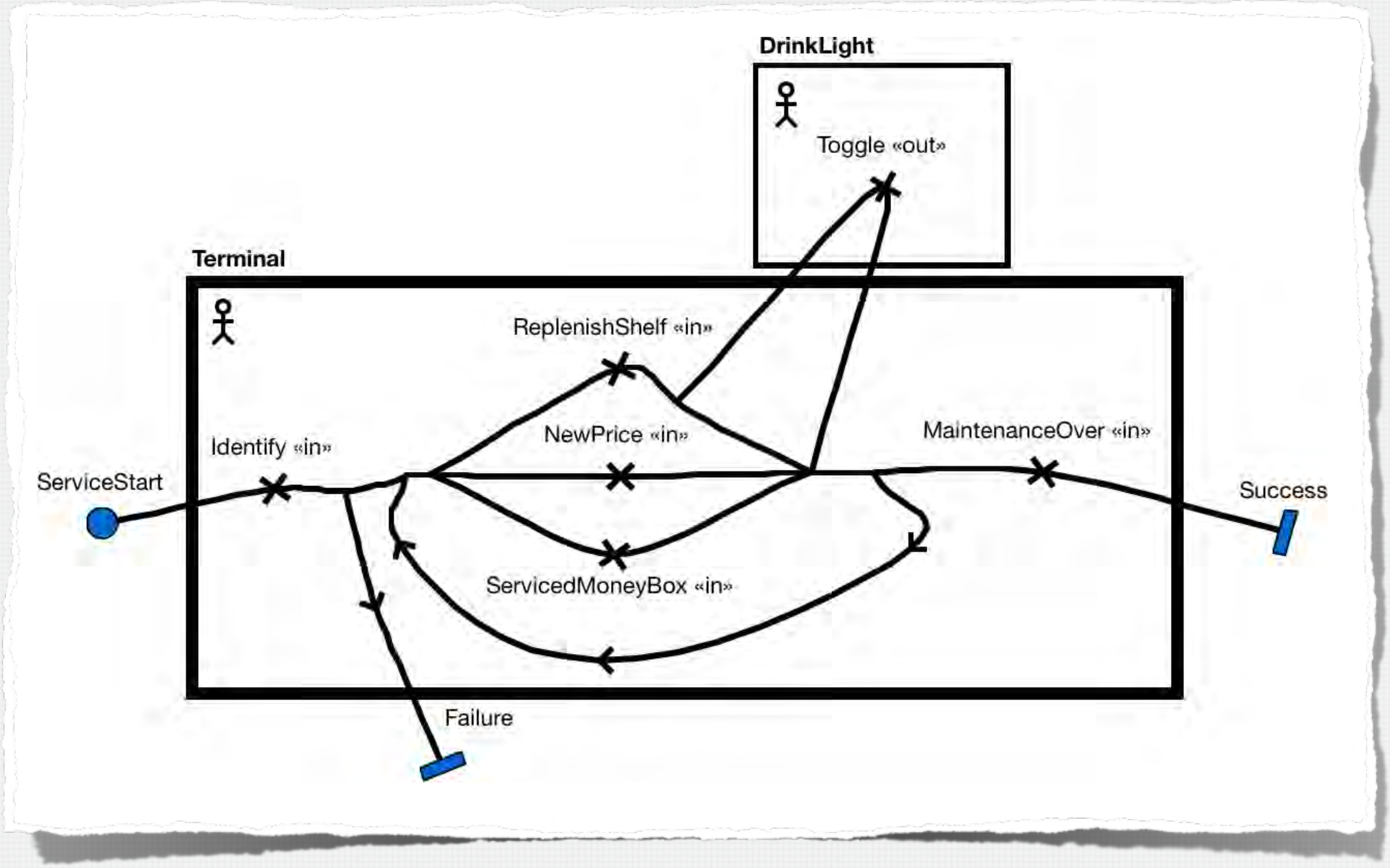
Drink Vending Machine EM



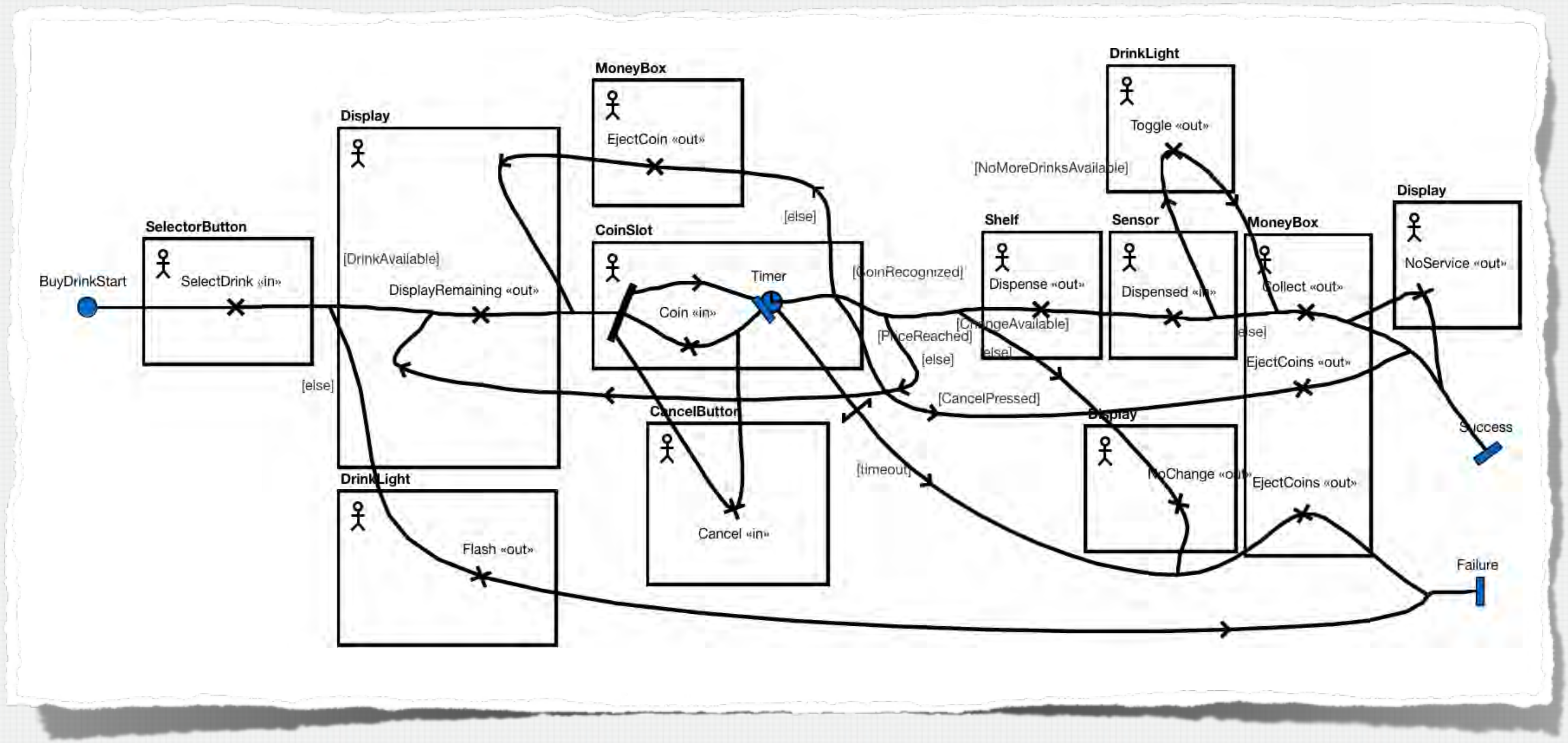
DVM EM Message Types

- Inputs
  - ServicedMoneyBox
  - NewPrice(s: Shelf, price: Money)
  - ReplenishShelf(s: Shelf, numberOfDrinks: Integer)
  - Identify(name: String, password: String)
  - Dispensed
  - SelectDrink
  - Cancel
  - Coin(c: Money)
- Outputs
  - DisplayRemaining(amount: Money)
  - NoChange
  - Dispense
  - Toggle(on: Boolean)
  - Flash
  - EjectCoins
  - EjectCoin
  - Collect(amount: Money)

DVM Protocol Model (1)



DVM Protocol Model (2)



BuyDrinkStart
SelectorButton
SelectDrink «in»
[else]
Display
[DrinkAvailable]
DisplayRemaining «out»
DrinkLight
Flash «out»
MoneyBox
EjectCoin «out»
[else]
CoinSlot
Coin «in»
Timer
[timeout]
CancelButton
Cancel «in»
[CoinRecognized]
[PriceReached]
[else]
[CancelPressed]
[ChangeAvailable]
[else]
Shelf
Dispense «out»
Sensor
Dispensed «in»
DrinkLight
Toggle «out»
[NoMoreDrinksAvailable]
MoneyBox
Collect «out»
[else]
EjectCoins «out»
EjectCoins «out»
Display
NoChange «out»
Display
NoService «out»
Success
Failure