Task Description

A modeller is responsible for editing several types of models such as Class Diagram, Sequence Diagram, State Diagram, Activity Diagram, and Use-Case Diagram. Assume that as a modeller, you are given a description of a system to enforce the parking policies at the Bauer complex, and your task is to update the State Diagram. Although your task is only to update/complete the State Diagram, you may find it necessary to also update the existing Class Diagram (i.e. adding a few model elements). You need to set the different parts of the transitions, namely, Triggering Event, Guard or Action, based on the description that is given to you for each part.

(See Next Page)

System Description: You are to create a UML State-Machine model of the proposed system to enforce the parking policies at the Bauer complex. Bauer provides parking for two types of users: customers and employees. There are two levels of parking: surface-level parking and underground parking. Surface-level parking is for customers only. They pay a $2 flat fee for parking. Employees park for free, but only if they park underground. Underground parking is also open to customers in the evenings, but of course customers must still pay ($2) to use it.

Gates control access to both levels of parking. Each gate controls traffic flowing in a single direction (e.g., traffic entering the complex), and it opens only for cars that approach the gate from the appropriate direction. Employees' cars have transponders that the gates can sense. Some gates open only for cars that have transponders. In addition, the system uses transponder data (IDs) to keep track of where employees' cars park. There are four gates that are controlled independently:

1) Gate **A** controls entry into the Bauer complex. It opens whenever it senses a car approaching from outside the Bauer complex. Cars that pass through gate A enter the surface-level parking lot.
2) Gate **B** controls access from surface-level parking to underground parking. Normally, it opens only if an approaching car has a transponder. But during evenings (18:00-6:00), it remains open to all cars.
3) Gate **C** opens whenever it senses a car approaching from the underground parking lot.
4) Gate **D** controls exits from the Bauer complex and collects parking fees. The gate opens whenever a $2 fee is paid (no change is provided). In addition, if an approaching car has a transponder, the gate opens if the car had parked underground (i.e., if the transponder triggered gate B since the car entered the complex).

Each of the gates closes 5 seconds after it opens, unless a blockage is detected, in which case it remains open and tries to close again after another 5 seconds. (Don't worry about modelling the delay of the gate opening and closing.)

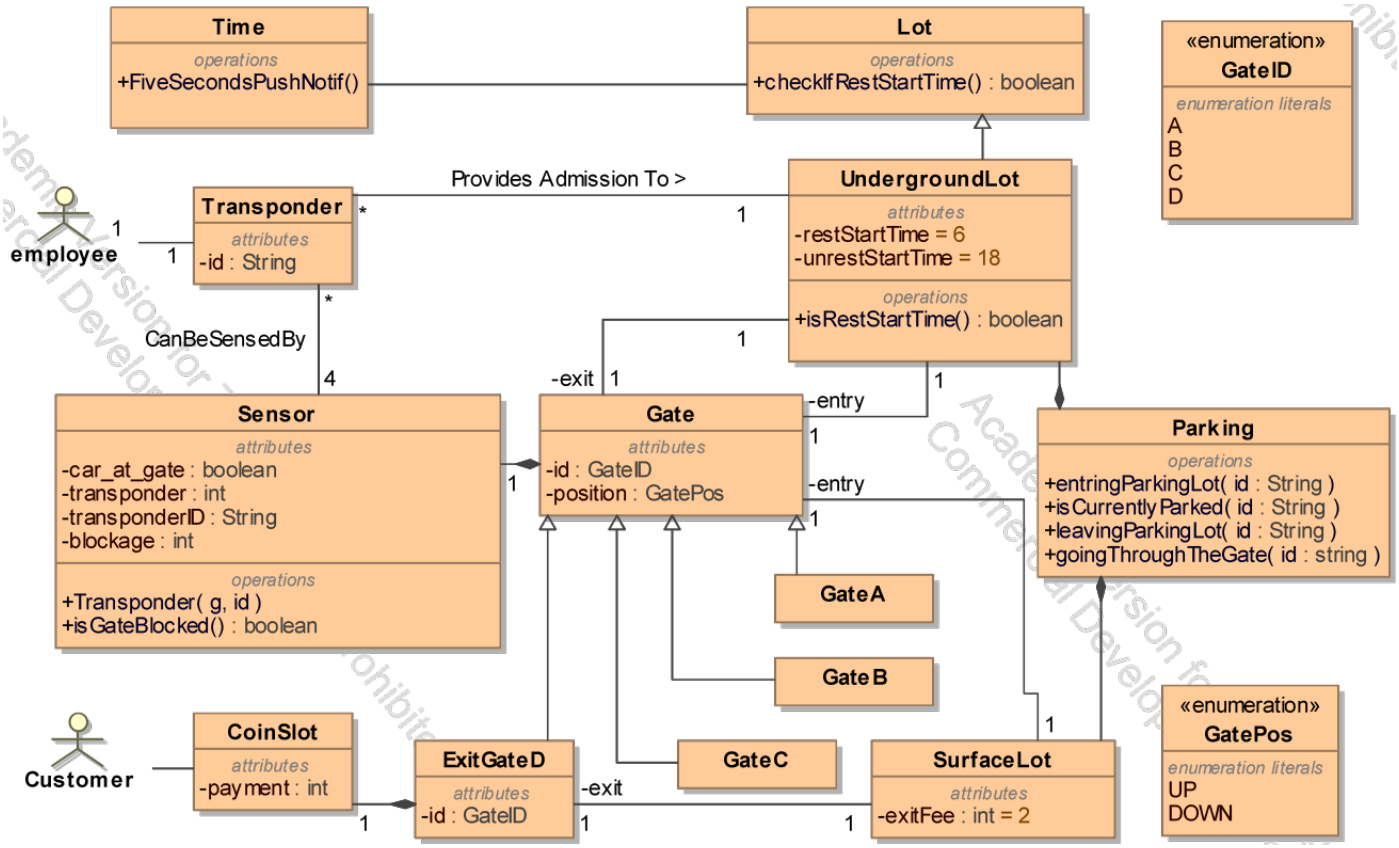

Fig. 1: Class Diagram for the Parking Lot application

Fig. 1 is the partial domain model for the system based on the interface choices already made. The events, conditions, and actions in your state-machine model should be expressed in terms of elements in the domain model. Moreover, the description of the domain model is given below. It helps you understand the domain, classes, attributes and the messages (operations) that you may need to know during the study. It is suggested that you take a brief look at the description before starting the tasks.

Lot: A general class representing the Lot concept.

- *checkIfRestStartTime:* Checks if a lot is in its restricted time period.

Time: Represents the time(er) concept.

- *FiveSecondsPushNotif:* Notifies the lots every five seconds to close the gate.

UndergroundLot: Represents the Underground parking of the parking lot.

- *restStartTime:* The Underground Lot has a restricted time for customers starting at 6:00am.
- *unrestStartTime:* The Underground Lot becomes unrestricted for customers starting at 18:00pm.
- *isRestStartTime:* Checks if the Underground Lot is in its restricted time period.

SurfaceLot: Represents the Surface-Level parking of the parking lot.

- *exitFee:* Represents the $2 parking fee that each customer must pay when exiting the lot.

GateID: Each Gate should have an ID of either A, B, C, or D. GateID is an enum variable (class) that represents the id of each gate.

GatePos: Gates can be either open or close, that is the gates' position can be either up or down. This enum class represents the allowable values for the gates' position.

Gate: Represents the Gates for both underground and surface-level parking lots.

- *id:* Each Gate has an id of the type GateID, that is a Gate can have an id of A, B, C, or D.
- *position:* Each Gate has a position attribute that is from the type of the class GatePos, and can have a value of up or down.

ExitGate: It shows the exit gate of the surface-level parking where the customer needs to put money in the coin slot.

CoinSlot: When exiting from the exit gate of the surface-level parking, the customer needs to pay the exit fee through a coin slot. The CoinSlot class represents the coin slot entity of the domain.

- *payment:* The amount that the customer has already inserted to the coin slot.

Sensor: Each Gate has a sensor the senses if a car is approaching the gate.

- *car_at_gate:* This is a boolean attribute that returns true if there is a car at the gate, and returns false if there is no car at the gate.
- *transponder:* Returns true if an approaching car has a transponder. Otherwise, returns false.
- *transponderID:* If an approaching car has a transponder, this attribute returns the ID of the transponder.
- *blockage:* Returns true in a case that a car has come through the gate, so that the gate does not close.
- *Transponder (g, id):* Returns true if a transponder (with id=id) is approaching gate **g** from the sensed direction.
- *isGateBlocked:* Returns true if the Gate is blocked.

Transponder: Each employee has a Transponder installed on their car. that provides admission to the underground parking and can be sensed by the sensors on the gates.

- *id:* Each Transponder has an ID.

Parking: This is a singleton class which records the states of cars with the transponder. Moreover, if a car enters to the parking, its transponder id should be recorded and be added to the collection of the parked cars' transponder ids. Also, when it leaves the parking, the car's transponder id must be removed from the list.

- *enteringParkingLot (id):* When a car is entering to the parking lot, this operation adds the transponder id of the car to the list of the transponder ids of the existing cars in the parking.
- *isCurrentlyParked (id):* This is a boolean operation which checks if a car with a transponder id had parked in the parking.
- *leavingParkingLot (id):* When a car is leaving the parking, the system uses this operation to remove the transponder id of the leaving car from the list of the transponder ids of the parked cars.
- *goingThroughTheGate (id):* Same as the enteringParkingLot operation.